Il tuo
primo
programma
in BASIC
Rodnay Zaks
SYBEX
GRUPPO
EDITORIALE
JACKSON

Il tuo primo programma in BASIC

Rodnay Zaks

GRUPPO
EDITORIALE
JACKSON

Sommario

Introduzione all'edizione italiana VII

Prefazione IX

Come leggere questo libro X

Che cosa imparerai XI

1

Parlare in BASIC 1

Programmazione 2

L'interprete BASIC 5

Cos'è il BASIC? 7

Quale BASIC? 8

Il tuo computer 9

Computer e sintassi 15

Comunicare con il tuo computer 17

L'uso della tastiera 18

Parlare in BASIC 24

Un programma più lungo 33

Riassunto 38

Esercizi 39

Far calcoli con il BASIC 41

Visualizzare i numeri 42

Notazione scientifica 43

Fare dell'aritmetica 44

Formati di stampa 47

Esempi applicativi 50

Riassunto 51

Esercizi 51

4

Memorizzare i valori ed usare le variabili 53

L'istruzione INPUT 54
I due tipi di variabile 57
Variabili stringa 60
Assegnare un valore ad una variabile 65
La tecnica del "contatore" 71
Riassunto 74
Esercizi 74

Scrivere un programma in modo chiaro 77

L'istruzione REM 78
Istruzioni multiple sulla stessa linea 80
L'uso degli spazi bianchi 81
Miglioramento della visualizzazione 83
Input abbreviato 84
La scelta dei nomi delle variabili 86
Una adatta numerazione delle linee 87
Riassunto 90
Esercizi 90

Prendere le decisioni 93

L'istruzione IF 94
Un esercizio di aritmetica 103
L'istruzione GOTO 108
L'istruzione IF rivisitata 111
Parliamo di nuovo dei contatori 112
Esercizio di aritmetica rivisitato 114
Controllare i dati in ingresso 115
Conversione miglia-chilometri 116
Compleanno 117

Riassunto 118

Esercizi 119

Le ripetizioni automatiche 121

La tecnica IF/GOTO 122

L'istruzione FOR...NEXT 126

Somma dei primi N numeri interi 129

Tabella di valori 130

Linee di asterischi 131

Cicli più avanzati 132

Caratteristiche addizionali 138

Riassunto 139

Esercizi 139

Creare un programma 141

Progetto dell'algoritmo 142

Costruzione di un diagramma di flusso 147

Codificazione 156

Eliminazione degli errori 158

Documentazione 160

Riassunto 162

Esercizi 163

Un esempio: conversione al sistema metrico 165

Progettiamo l'algoritmo 166

Diagramma di flusso 166

Codificazione 175

Collaudo 184

Riassunto 186

Esercizi 187

10 Il prossimo passo 189

Cosa puoi fare con il BASIC 190

Migliorare la tua capacità 190

Ancora BASIC 192

Conclusione 195

Appendici

Risposte ad alcuni esercizi 197

Le più comuni parole riservate del BASIC 203

Glossario del BASIC 204

Indice Analitico 210

Introduzione all'edizione italiana

L'autore di questo libro, R. Zaks, è già conosciuto, in Italia, nell'ambito degli utenti di personal computers, per le sue opere; specialmente per i volumi molto apprezzati sulla programmazione in linguaggio assemblativo dei microprocessori Z80 e 6502, già editi dal Gruppo Editoriale Jackson.
In questo libro, Zaks si prefigge di insegnare, senza alcun formalismo inutile, le nozioni fondamentali del BASIC a chiunque ne abbia la volontà e possieda, o possa utilizzare, un piccolo computer.
Attraverso un metodo simile a quello usato per l'apprendimento delle lingue, l'autore mette in grado chiunque, "tra gli 8 e gli 88 anni" anche completamente digiuno di nozioni tecniche, di programmare in linguaggio BASIC dopo pochissime ore. Il libro, con un dialogo non formale ed un aspetto non certo tradizionale per i testi tecnici, cerca di sdrammatizzare quello che è stato finora un argomento accessibile solo a pochi: la programmazione di un computer.

Aprire questo libro è un po' come sfogliare il testo di lingua straniera delle scuole medie e l'insegnante (diciamo, piuttosto, l'interlocutore), invece della professoressa di lingue, è un computer molto più docile e paziente.
Un punto essenziale per trarre vantaggio da questo libro (ma la cosa è altrettanto valida per qualsiasi testo di programmazione) è la disponibilità di un computer. (Va bene anche il più piccolo e meno costoso in commercio).
Senza questa disponibilità nessun libro, corso o enciclopedia possono insegnarti a diventare un buon programmatore. È come voler parlare una lingua straniera, solamente leggendo il testo di grammatica!

Oltre alla disponibilità di un computer, è necessario possedere una dote per poter scrivere programmi con successo: la disciplina. Solamente seguendo rigorosamente le regole imposte dalla sintassi ed organizzando precedentemente il proprio lavoro (questo libro ti spiegherà il come ed il quando), si possono ottenere risultati soddisfacenti e delle soddisfazioni.
Vediamo ora le difficoltà alle quali potrebbe andare in contro il lettore di lingua italiana rispetto al lettore di lingua inglese.

La conoscenza dell'inglese non è indispensabile per entrare nel mondo dei computer, anche se può essere d'aiuto, specialmente all'inizio, per poter ricordare e capire i termini tecnici. Infatti tutte le parole dei linguaggi programmativi e la maggior parte dei termini tecnici sono in lingua inglese e la loro traduzione in italiano, spesso, perde la concisione e l'efficacia del termine originale. Per questo motivo, nel testo ho lasciato in Inglese tutti i termini che vengono solitamente usati in lingua originale (ho fornito, di solito tra parentesi, la traduzione letterale per evitarti di consultare il vocabolario). Ti consiglio, comunque, di non cercare di ricordare il termine inglese attraverso la sua traduzione letterale, ma di ricordarlo piuttosto attraverso il suo significato materiale (quale azione compie).

L'uso del punto decimale al posto della virgola.
Anche se ormai tutti lo sanno (a causa della diffusione delle calcolatrici tascabili) ritengo opportuno ricordare che, nella notazione anglosassone e perciò nell'ambito della programmazione, il punto decimale sostituisce la nostra virgola nei numeri decimali. (Per esempio, 25.60 è l'equivalente del nostro 25,60). Voglio ricordare per completezza che non si può usare il punto per dividere di tre in tre le cifre. (Dieci milioni va scritto 10000000 e non 10.000.000).
A questo punto, sono certo che chiunque sia riuscito a leggere queste mie note fin qui, diventerà sicuramente, per la sua pazienza, un buon programmatore.
Sperando che questa mia traduzione ti renda piacevole la lettura di questo libro, ti auguro un proficuo e divertente approccio alle "basi" del linguaggio BASIC.

Mauro Monsignori

Prefazione

Sono già stati scritti centinaia, forse migliaia, di libri sul BASIC; perchè farne un altro? Semplicemente perchè è cambiato il genere dei lettori. Nel passato, l'uso dei linguaggi programmativi era un privilegio dei pochi che avevano accesso ad un computer: i programmatori erano una piccola elite. Ora la situazione è cambiata, i personal computers hanno fatto del BASIC il linguaggio più usato e più accessibile mai esistito e, oggigiorno, la maggioranza degli utenti di computers hanno una preparazione tecnica insignificante o nulla. Essi usano il computer nel tempo libero, nella scuola, negli affari o nella loro professione.
Questo libro è indirizzato a questi nuovi utenti: esso non soltanto *sembra* diverso, ma *lo è* realmente. È rivolto ai principianti e perciò presume che il lettore non abbia alcuna conoscenza tecnica. Il libro è rivolto a chiunque, tra gli 8 e 88 anni, desideri imparare rapidamente a fare i primi passi con il BASIC.
L'autore ritiene che tutti i nuovi utenti di computer, che vogliono imparare a scrivere i loro programmi in BASIC, siano pieni di entusiasmo e giovani o, almeno, giovani nello spirito. Essi vogliono un accesso immediato, semplice ed educativo all'apprendimento del BASIC e questo è il metodo usato in questo libro: esso è stato scritto per rendere lo studio del BASIC facile e divertente.
Inoltre, questo libro mira ad insegnarti, in poche ore, i concetti essenziali del BASIC: in un'ora dovresti essere capace di scrivere il tuo primo programma in BASIC e, dopo poche altre, essere già in grado di scrivere programmi utili e significativi.
Il tempo sta passando ... incominciamo.
L'autore ti augura un buon viaggio lungo il magico sentiero della conoscenza.
Rodnay Zaks Berkeley, Gennaio 1983

Come leggere questo libro

Questo è un libro didattico, perciò dovrai leggere un capitolo dopo l'altro e capire bene quello letto, prima di passare al successivo. Ho fornito degli esercizi, alla fine di ogni capitolo, per aiutarti a valutare il tuo nuovo livello di abilità. Fanne quanti più ti è possibile. Le risposte ad alcuni esercizi sono nell'Appendice A alla fine del libro.
Se hai un computer, prova tutti i programmi, perchè, per imparare veramente e poi ricordare, devi fare pratica ed esperienze. Questo libro ti fornirà le nozioni pratiche e le conoscenze di cui hai bisogno per iniziare, ma ricorda che nulla può sostituire l'esperienza personale.

Lo scopo finale di questo libro è di farti iniziare a programmare in BASIC rapidamente ed in modo efficace. Per raggiungere questo scopo e per renderti il cammino facile, ho dovuto fare alcune scelte: perciò questo libro non descrive tutte le caratteristiche ed i concetti del BASIC, ma solo quelli più importanti.
La mia speranza è che, attraverso questo libro, tu possa capire ogni cosa rapidamente, possa essere in grado, senza indugi, di scrivere il tuo primo programma in BASIC e possa, divertendoti, apprezzare la potenza della tua nuova abilità di programmatore.

Che cosa imparerai

Il primo capitolo spiega il linguaggio dei computers e ti presenta gli eroi di questo libro: il Computer, l'Interprete, il Programma, le Istruzioni e gli altri personaggi del cast.

Il secondo capitolo ti mostra come comunicare con il tuo computer, utilizzando le risorse della tastiera e del video. Imparerai a scrivere sulla tastiera il tuo primo programma in BASIC e ad eseguirlo.

Il terzo capitolo ti mostra come eseguire i calcoli con il BASIC.

Il quarto capitolo ti aiuta a scrivere programmi che possano essere usati ripetutamente. Inoltre imparerai ad usare correttamente ed efficientemente le variabili.

Il quinto capitolo ti mostra come rendere i tuoi programmi chiari e leggibili.

Il sesto capitolo ti mostra come prendere complesse decisioni basate sulla logica e sui valori.

Il settimo capitolo spiega come automatizzare le funzioni ripetitive, usando nel programma i *cicli.*

L'ottavo capitolo mostra il metodo corretto per scrivere un programma partendo dall'algoritmo fino al programma documentato e funzionante (compresa la compilazione del diagramma di flusso).

Il nono capitolo ti aiuta ad applicare tutti questi concetti ad un caso pratico.

Il decimo capitolo ti aiuta ad esaminare il passo successivo da fare per migliorare la tua abilità di programmatore.

Le Appendici A, B e C offrono le risposte ad alcuni esercizi, una lista delle parole riservate del BASIC e un glossario.

Ora se sei pronto, apriamo l'album di famiglia perchè ti presenterò gli eroi di questo libro

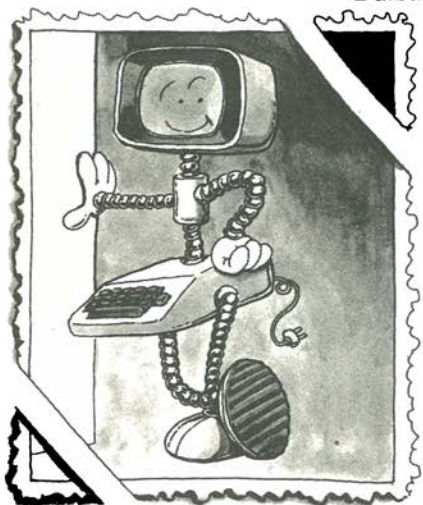
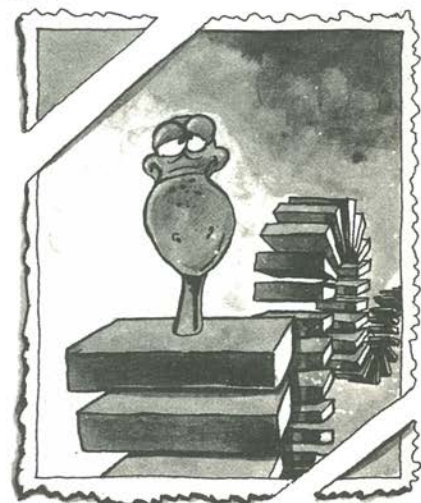
Conosciamo i nostri eroi

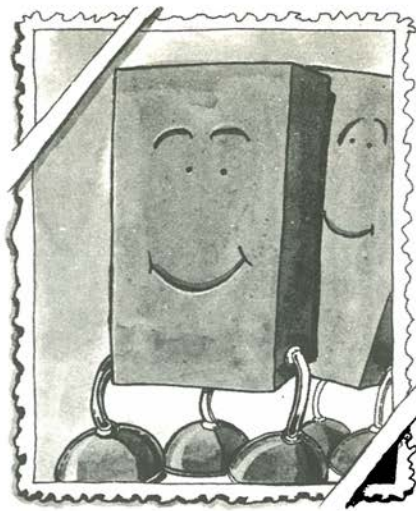
Personaggi (in senso orario): Dino il programmatore, l'Interprete BASIC in piedi sul suo amico il Computer, Snake (serpente) il Programma, Bug, il malefico difetto, due Variabili, le Istruzioni di programma pronte a marciare verso i loro punti assegnati e Flowchart, l'indispensabile diagramma di flusso sul quale Dino si riposa. Ahi, il nostro Bug sembra essere dappertutto!

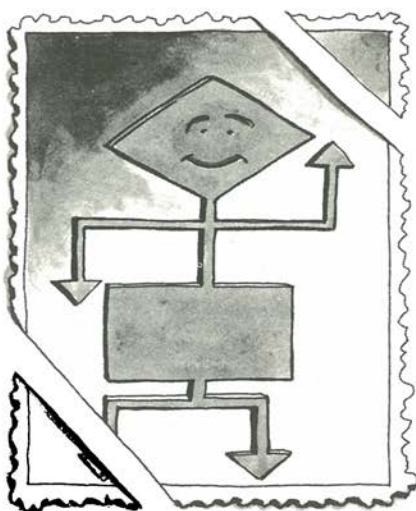
Peccato ... il nostro malefico Bug ha colpito ancora. Basta una foto di gruppo e ...

Questo è l'Interprete BASIC. Quando è sveglio, risiede nella memoria del tuo Computer. Il suo lavoro è tradurre le tue istruzioni al computer. Egli cercherà in tutti i modi di aiutarti.

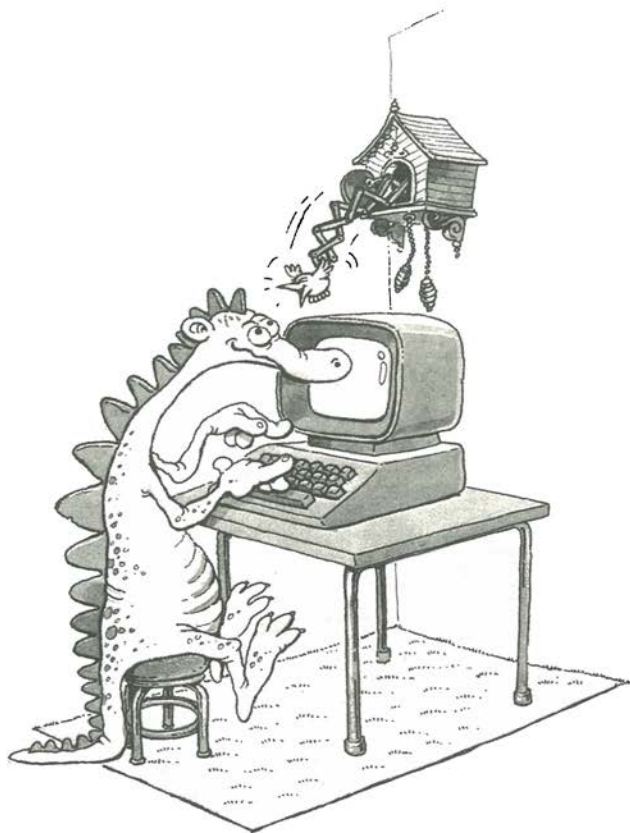
Non dimenticare mai la sua faccia. Questo è Bug, il difetto. Ti renderà la vita difficile; perciò fai del tuo meglio per tenerlo lontano dai tuoi programmi.

L'album di famiglia

Questo è il tuo amico il Computer ... al tuo servizio.

No, non è un mostro ... è Snake, il programma. È fatto di istruzioni che imparerai a comporre. È molto docile, una volta che lo conoscerai, però tieni Bug lontano da lui.

Questo è Dino. È molto simpatico e, sebbene non abbia una educazione formale, ti mostrerà quanto è semplice scrivere i programmi in BASIC.

Ecco le istruzioni BASIC, pronte a congiungersi con Snake il programma.

Questa è una variabile numerica. Il suo vestito è contrassegnato con il suo nome ed ha un valore impresso. È infelice, perchè vorrebbe tornare al suo posto riservato nella memoria.

Questo è il tuo miglior amico, Flowchart il diagramma di flusso. Ti aiuterà a progettare programmi che funzionino.

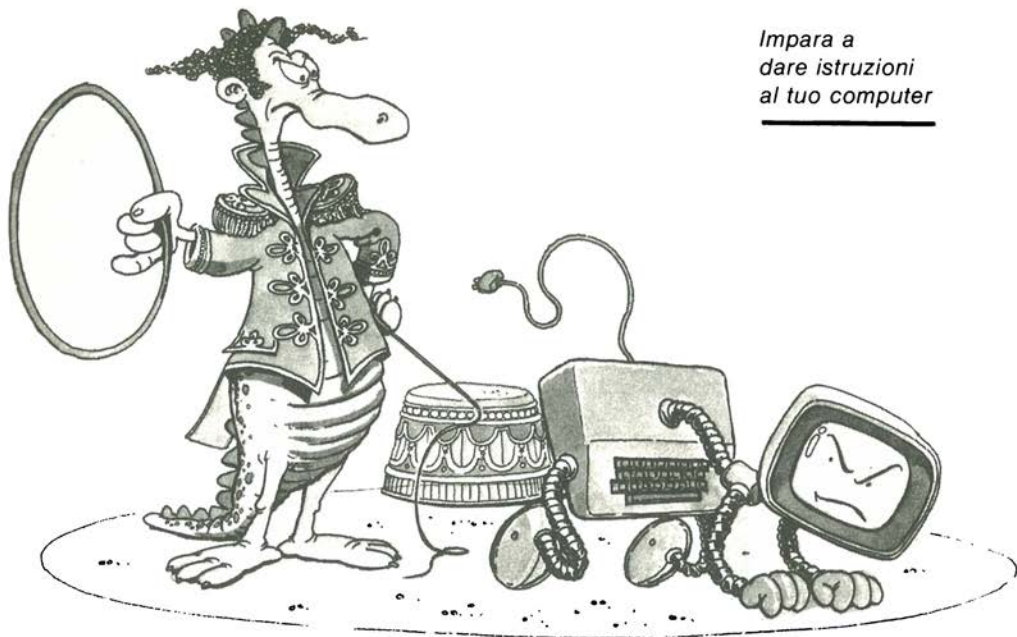
Capitolo 1

Parlare in BASIC

Ho dichiarato nella prefazione che "sarai in grado di scrivere il tuo primo programma in BASIC entro un'ora" e, allora, perchè iniziare con un capitolo di concetti e definizioni? Non stiamo forse perdendo tempo? Al contrario: il nostro scopo è di imparare e ricordare le nozioni e, un vero apprendimento, richiede una profonda conoscenza.

Le nozioni presentate in questo capitolo ti aiuteranno a capire meglio che cosa è la programmazione, come viene eseguito un programma BASIC e qual'è il vocabolario dei computers. Prima di iniziare a scrivere il tuo primo programma, ci sono alcune importanti definizioni e concetti che dovrai imparare. Una volta che avrai capito questi vocaboli, potrò spiegarti, in un modo semplice e preciso, *cosa accade e cosa fare* **e, a tal punto, dovresti essere capace di seguire il tutto facilmente. Leggi questo capitolo attentamente e preoccupati di capire veramente quello che stai facendo e ... non soltanto di premere i tasti! Inizieremo ad imparare come fornire le istruzioni al computer: questa è la** *programmazione*, **poi spiegheremo la necessità dei** *linguaggi programmativi*, **come il BASIC ed, in seguito, chiariremo che cosa sia un** *interprete BASIC*. **Esploreremo la storia di questo linguaggio, dei suoi** *dialetti* **ed il loro uso. Per finire, esamineremo i componenti di un** *computer* **ed impareremo alcuni termini del gergo tecnico usato per descrivere questi componenti.**

Programmazione

Il tuo computer è una macchina progettata per elaborare le *informazioni* sia numeriche che di testi. Per esempio, puoi far visualizzare sullo schermo del tuo computer sia parole e frasi (questa è chiamata *elaborazione di testi*) che convertire un peso espresso in once al suo valore in grammi (questa è chiamata *elaborazione numerica*). Affinchè il tuo computer esegua queste elaborazioni, è necessario impartire le istruzioni in una forma o "linguaggio" a lui comprensibile.
Ogni computer può "capire" (cioè riconoscere ed eseguire) solamente un piccolo numero di istruzioni (diciamo, poche decine).
Le istruzioni che un computer può interpretare direttamente sono chiamate istruzioni in *linguaggio macchina*. Queste istruzioni sono memorizzate in forma *binaria*, cioè in gruppi di 1 e 0, nella memoria del computer. Ogni 0 e 1 è detto un *bit* ed ogni gruppo di 8 di bits è detto *byte*.

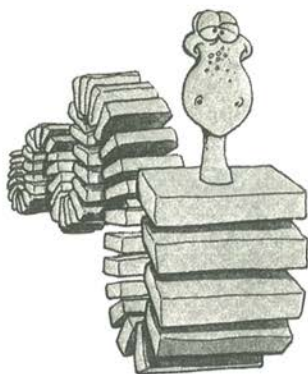
Impara a dare istruzioni al tuo computer

Una sequenza di istruzioni che compia un qualcosa di utile è detto *programma*. (Una sequenza che non faccia niente è un *errore*). Il tuo computer elabora un programma eseguendo una istruzione dopo l'altra.
Sfortunatamente, scrivere un programma (una sequenza di istruzioni) in linguaggio macchina, cioè in forma binaria, è un procedimento lento e noioso.
Idealmente, ci piacerebbe poter dare e scrivere i comandi nella lingua di tutti i giorni (diciamo, in Inglese) ad un computer che li esegua. Ma ciò non è possibile, perchè un computer non è in grado di capire niente della nostra lingua parlata e scritta. La ragione di ciò è molto semplice: un computer esegue gli ordini in modo rigoroso ed esatto; è logico e preciso e richiede istruzioni chiare e non ambigue in una certa sequenza e forma. Il problema con una lingua parlata risiede nella lingua stessa; le frasi possono essere ambigue e spesso il loro significato può dipendere dal contesto come un gesto o una espressione del viso. Questo tipo di comunicazione non può essere interpretata dal computer.

Anche se attentamente scritta, una lingua di tutti i giorni resta insufficientemente precisa per un computer. Per esempio, non si può dire ad un robot computerizzato "va in cucina e cuoci un uovo" ed aspettarsi un risultato, a meno che il robot non sia stato programmato per muoversi nella tua cucina. Esso deve essere istruito (o programmato) per muoversi in un ambiente come il nostro e, anche se il robot fosse stato addestrato a muoversi nella *tua* cucina, non saprebbe districarsi in quella di un tuo amico, perchè gli oggetti sono sistemati in maniera diversa.
Quindi ricordati che la comunicazione con un computer deve essere chiara, precisa e non ambigua.

È per questo motivo che sono stati inventati dei "linguaggi" semplificati per poter comunicare con i computers. Ricordati che il linguaggio binario è il più semplice da capire per un computer, ma è difficilmente trattabile dall'uomo. Perciò sono stati inventati altri linguaggi per facilitare la comunicazione. Questi linguaggi sono simili all'Inglese comune e sono chiamati *linguaggi ad alto livello.*
Per una chiara ed efficace comunicazione con il computer possono essere usate, come comandi predefiniti, solo poche parole di Inglese e, inoltre, le frasi (chiamate *istruzioni o statements*) devono obbedire a rigorose regole grammaticali; l'insieme delle quali è chiamata *sintassi* del linguaggio. La combinazione del ristretto vocabolario e della sintassi è chiamata *linguaggio programmativo.* Il BASIC è uno di essi.
Per concludere, un *linguaggio programmativo* è un insieme di regole (la sintassi) e parole e simboli (il vocabolario) che ti permettono di inviare le istruzioni ad un computer in una forma a lui esattamente comprensibile.
Una sequenza di tali istruzioni è detta *programma.*
Ecco un esempio: supponiamo di voler sommare 2+2 e visualizzare il risultato. Usando il BASIC dovremmo scrivere:

```
1  R = 2+2
2  PRINT R
```

dove R sta per "risultato". (PRINT significa "stampa")

*"Io amo il Basic.
Parlami in BASIC".*

*"Ti ricordi di me?
Io sono il programma
e sono fatto di istruzioni".*

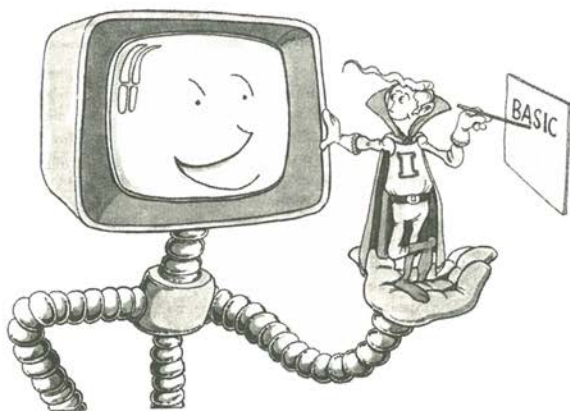
Ma aspetta ... abbiamo detto precedentemente che l'unico linguaggio che un computer è in grado di capire direttamente è il *linguaggio macchina* e noi abbiamo fornito al computer istruzioni in un linguaggio simile all'Inglese. Non c'è una contraddizione?

No, non c'è nessuna contraddizione. Infatti il semplice computer non può capire il BASIC direttamente, o per la stessa ragione, ogni altro *linguaggio programmativo ad alto livello* (cioè un linguaggio che usi frasi in Inglese). Ma, per essere capito da un computer, un programma scritto in un linguaggio ad alto livello, come il BASIC, deve essere *interpretato* da uno speciale programma, chiamato appunto *interprete*. In altre parole, tu parli in BASIC al tuo computer tramite un interprete. Dunque, per poter eseguire un programma BASIC, il tuo computer deve avere un interprete BASIC. Impariamo, ora, cosa fa un interprete.

L'interprete BASIC

"Ti ricordi di me? Io sono l'interprete BASIC, pronto a tradurre le tue istruzioni al computer. Sono un programma e risiedo nella memoria del tuo computer".

Un interprete BASIC legge ogni istruzione BASIC che scrivi sulla tastiera e la traduce automaticamente in una sequenza di istruzioni in linguaggio macchina che il tuo computer può capire ed eseguire. Questo procedimento ti è completamente invisibile (cioè, ha luogo dentro il computer). Una volta attivato sul tuo computer il programma interprete, per tutti gli scopi pratici il tuo computer può parlare in BASIC. Il tuo computer può essere in grado di parlare anche altri linguaggi programmativi, purchè sia provvisto del relativo interprete.
Esistono molti tipi di interpreti BASIC che puoi usare sul tuo computer ed ora ne descriveremo i due tipi principali: *residente* e *non-residente*.
L'interprete BASIC *residente* è quello esistente in quasi tutti i piccoli computer. Esso è chiamato così perchè "risiede" permanentemente nella memoria del computer ed è subito accessibile non appena il computer viene acceso o dopo un comando tipo B o BASIC. Non appena il simbolo di conferma del BASIC (chiamato *prompt* (>)) appare sul tuo schermo, tu sai che il computer è pronto ad eseguire le tue istruzioni in BASIC.

Un interprete BASIC residente generalmente ha un inconveniente: può fornire solo una versione minima del BASIC. Poichè un interprete residente è "costruito" permanentemente nella memoria interna del computer, esso deve occupare poco spazio, perchè la dimensione totale della memoria è limitata. La memoria del computer deve contenere il programma, l'interprete BASIC ed avere sufficiente spazio per i calcoli, l'organizzazione del sistema ed i dati sui quali si opera. Questo fabbisogno di spazio limita la dimensione e, perciò, restringe la complessità dell'interprete residente. Sui computer che possiedono una piccola quantità di memoria, l'interprete BASIC è spesso un "tiny (minuscolo) BASIC" che impone dei limiti a ciò che ci puoi fare.

"Quando mi trovo nella memoria, il tuo computer parla in BASIC".

Tutti i BASIC residenti sono, comunque, sufficienti per imparare a programmare in BASIC, almeno per le intenzioni di questo libro. Ma più in là, quando imparerai a scrivere dei programmi più complesi, probabilmente sentirai il bisogno di avere più funzioni e quello che vorrai sarà un interprete *non residente*.
Per possedere più funzioni, l'interprete deve essere più complesso e, perciò, di dimensioni più ampie. Esso, per questo motivo, viene memorizzato in un dispositivo di memoria di massa, come una cassetta o un dischetto, per non occupare sempre la maggior parte della memoria disponibile.
Sulla maggior parte dei piccoli computers, è generalmente necessario acquistare una memoria aggiuntiva per poter usare un interprete più grande.
Un interprete più potente fornirà una versione del BASIC chiamata, a seconda del fornitore, "*full BASIC*" (BASIC completo), *"extended BASIC"* (BASIC esteso), *"floating-point BASIC* (BASIC a virgola mobile) o *"advanced BASIC" (BASIC avanzato)* o, a seconda della memoria di massa usata, *"cassette BASIC"* o *"disk BASIC"*. Generalmen-

te, tutti i programmi scritti con un BASIC residente funzioneranno senza alcun cambiamento anche con un "full BASIC". In questo caso, le due versioni hanno una *compatibilità upwards* cioè sono compatibili in senso ascendente.
Per usare un BASIC su cassetta o su disco, devi prima trasferire l'interprete dalla cassetta o dal disco nella memoria del computer (operazione chiamata *caricamento (load)* dell'interprete), poi devi dare uno specifico comando tipo FBASIC, per attivare il tuo "full BASIC".
L'interprete visualizzerà poi il prompt, il quale è generalmente diverso da quello dell'interprete residente proprio per evitare confusione. Solamente a questo punto puoi introdurre le istruzioni BASIC.
Spieghiamo, ora, cosa è il BASIC, come fu inventato e i dialetti che ne sono derivati.

Cos'è il BASIC

I linguaggi ad alto livello sono stati inventati affinchè l'utente possa fornire le istruzioni al computer, cioè programmarlo, più facilmente. Durante questi anni, sono stati inventati centinaia di linguaggi programmativi.
In un primo tempo i computers furono usati principalmente per scopi scientifici ed i primi linguaggi programmativi furono studiati per facilitare il calcolo numerico. Infatti il "nonno" dei linguaggi, il FORTRAN (FORmula TRANslator, cioè traduttore di formule), era stato studiato specificatamente per risolvere calcoli numerici. Il FORTRAN, comunque, risentiva di molti inconvenienti e, perciò, furono studiati altri nuovi linguaggi. Il BASIC era uno di questi; il COBOL, l'APL ed il Pascal erano gli altri che furono largamente usati.
L'invenzione del BASIC ha rappresentato il maggior avvenimento; esso era stato studiato per essere semplice e facile da imparare; inoltre era *interattivo*.
Vediamo cosa significa BASIC.
BASIC significa **B**eginners **A**ll-purpose **S**ymbolic **I**nstruction **C**ode, cioè *Codice ad istruzioni simboliche per principianti, adatto a tutti gli usi.* Fu inventato nel 1964 da John Kemeny e Thomas Kurtz al Dartmouth College, lavorando con una sovvenzione della National Science Foundation. Il traguardo degli autori era di studiare un linguaggio che potesse essere usato facilmente da un principiante. Essi hanno raggiunto lo scopo: oggigiorno, il BASIC è uno dei linguaggi più facili da imparare.
Poichè il BASIC è nato come linguaggio interattivo (infatti fu il primo linguaggio interattivo) un utente può interagire col programma attraverso un terminale, piuttosto che immettere pile di schede perforate IBM come negli altri vecchi linguaggi. Il BASIC, originariamente, funzionava su un sistema GE225 in time-sharing (con accesso a suddivisione temporale) al Dartmouth College dove alcuni terminali erano accessibili in diversi luoghi dell'università e molti utenti potevano accedere al computer simultaneamente.

Il successo del BASIC fu rapido, tanto che la General Electric (GE) decise di sfruttarlo subito commercialmente.
Kemeny e Kurtz pubblicarono il primo libro sul BASIC nel 1967 e la Hewlett Packard (HP) e la Digital Equipment Corporation (DEC) decisero di rendere il BASIC disponibile sulla maggior parte dei loro computers.
Il BASIC offre due vantaggi principali rispetto ad altri linguaggi come il FORTRAN:

1. A vantaggio dell'utente: il BASIC è il più facile da imparare, *specialmente* per un principiante.

2. A vantaggio del costruttore: il BASIC è il linguaggio più facilmente installabile su un computer perchè l'interprete è semplice ed occupa poca memoria.

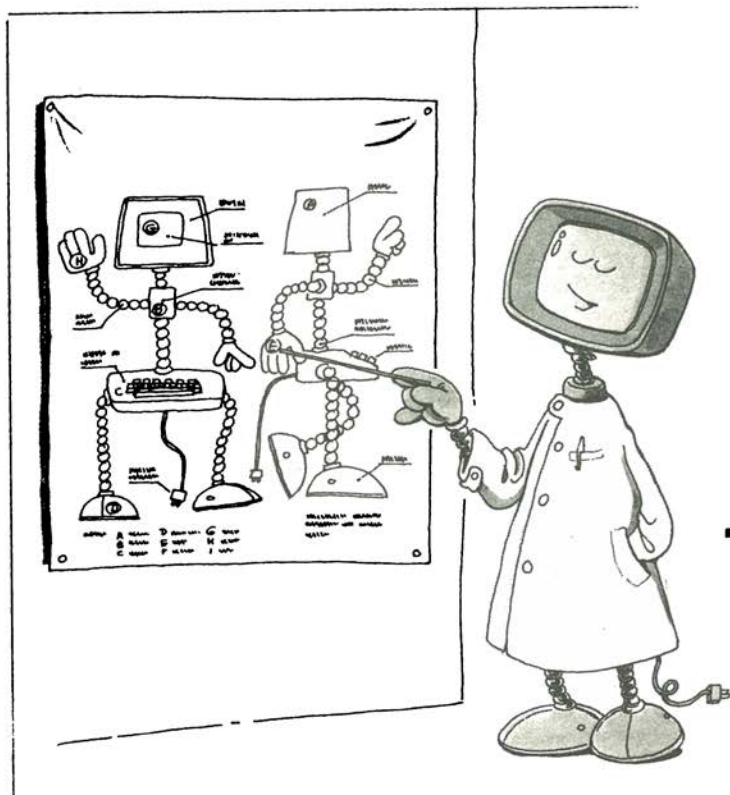
Un terzo fattore che ha contribuito all'enorme successo del BASIC è stato l'avvento dei microcomputer a basso costo. Quando negli anni settanta i microcomputers divennero largamente accessibili, il BASIC divenne il linguaggio di programmazione universale in questi piccoli computers.
Poichè l'interprete, per una versione semplificata di BASIC, richiede solamente 4K (4096 bytes) di memoria, anche i più piccoli computers potevano alloggiare un BASIC residente. (Ricordati che BASIC residente significa che l'interprete è alloggiato nella memoria permanente del computer). I più recenti computers hanno dimensioni di memoria molto più grandi (64K o più, dove 1K equivale a 1024 bytes) e possono essere muniti di versioni molto potenti di BASIC.
Oggi, il BASIC è usato in quasi tutti i computers. Con il passare degli anni, i costruttori hanno aggiunto estensioni e nuove funzioni al linguaggio, tanto che il BASIC, oggi, è il linguaggio meno standard che esiste: non ne esistono due versioni uguali. Il BASIC è diventata una famiglia di linguaggi piuttosto che un linguaggio unico.
Sebbene siano stati proposti molti standard, nessuno ha mai avuto successo e, a tutt'oggi, nessuno sembra poterne avere.
Questo significa, forse, che bisognerà studiarsi di nuovo il BASIC su ogni nuovo computer? Non precisamente. Una volta che avrai imparato i concetti fondamentali, comuni a tutte le versioni, potrai facilmente studiarti le estensioni che ogni versione offre. Tutti i BASIC hanno essenzialmente lo stesso nucleo di istruzioni ed imparerai molto di più su di esse, continuando a leggere questo libro.

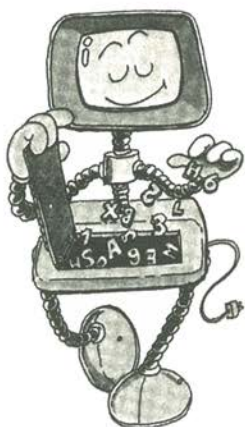
Quale BASIC?

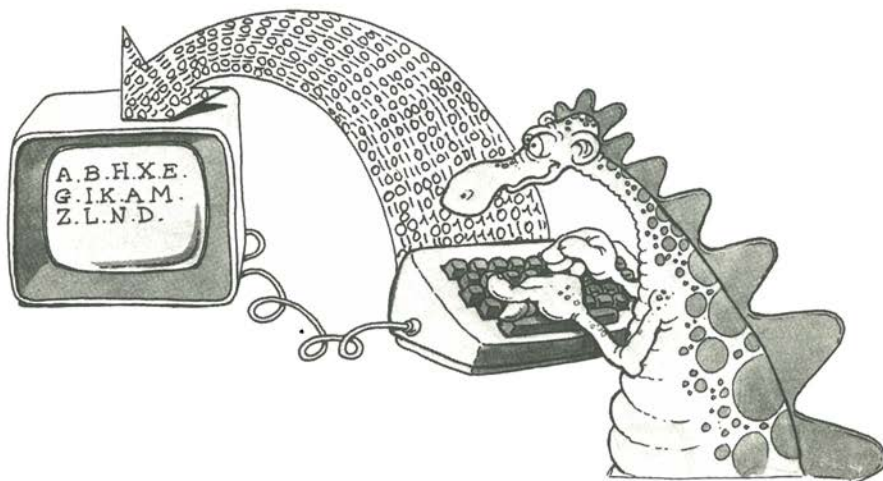
Il tuo computer ha, probabilmente, disponibili varie versioni *diverse* del BASIC. Precedentemente ne abbiamo descritte di due tipi: BASIC residente (normalmente un mini-BASIC) e BASIC non residente (un BASIC completo o esteso). Esaminiamo brevemente alcune loro differenze.

Un "mini" o "tiny BASIC" ha meno proprietà e funzioni rispetto ad un "full" o "extended BASIC". Una usuale limitazione di un "mini-BASIC" è di poter operare solo su numeri interi, cioè non poter trattare numeri frazionari.
Una tale versione è detta "integer BASIC" (intero), in contrasto con una versione migliorata che tratta numeri frazionari chiamata "floating-point BASIC" (a virgola mobile). Quest'ultima capacità è fortemente raccomandata a chi prevede di fare dei calcoli.
Un mini-BASIC generalmente memorizza le informazioni su una cassetta piuttosto che su un disco e generalmente, memorizza i programmi ma non i dati (non tratta i "files"). Normalmente, esso può trattare solo informazioni che siano parte del programma o che siano immesse dalla tastiera. Per contrasto a quanto detto, un BASIC completo usato con le unità a disco, offre grossi vantaggi e potenza nel trattamento di quantità di dati (files) e nell'elaborazione dei programmi.
Alcuni costruttori forniscono versioni ancora più avanzate che sono specifiche della ditta costruttrice. Gli utenti sono generalmente avvertiti di questa capacità dalla dizione "extended BASIC"; ciò significa che sono disponibili per il programmatore evoluto risorse maggiori e più potenti. Comunque, l'utilizzazione di queste speciali risorse rende, generalmente, il BASIC incompatibile con gli altri interpreti BASIC.
Il BASIC usato in questo libro è un mini-BASIC universale così che potrai mettere in pratica le esperienze fatte su tutte le versioni del BASIC. Puntualizzeremo, comunque, nel testo le variazioni e le estensioni più comuni.
Per concludere, non devi preoccuparti della versione che stai utilizzando, perchè, in seguito, se desidererai scrivere programmi più complessi, potrai studiarti il manuale di riferimento della versione del BASIC che stai usando. Inoltre nell'ultimo capitolo verranno presentate molte caratteristiche del BASIC esteso.
Ora che ne sappiamo di più sui linguaggi programmativi in generale e sul BASIC, in particolare, cerchiamo di imparare qualcosa sul tuo computer e su come elabora le informazioni.

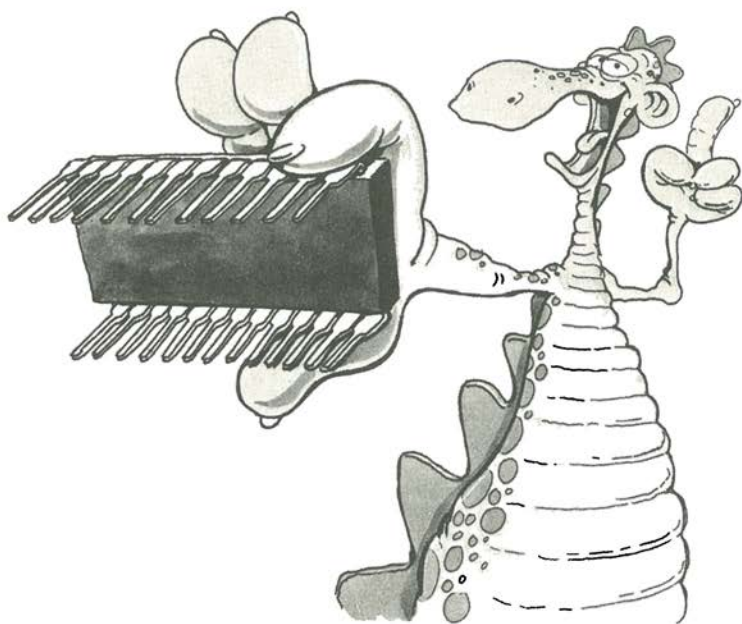
Il tuo computer

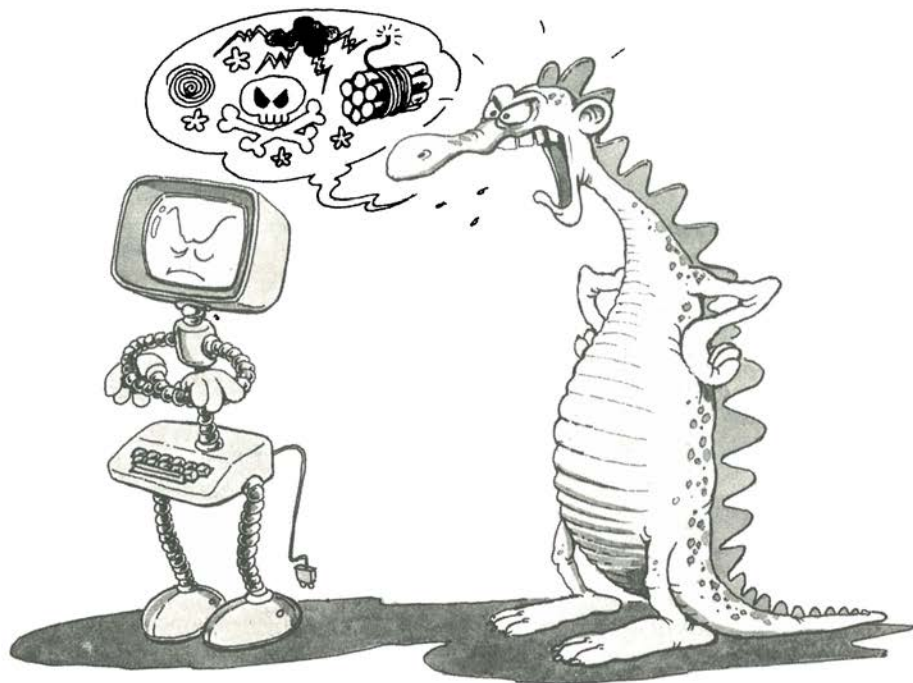
Il tuo computer elabora informazioni e comunica con te attraverso una tastiera ed uno schermo, e, qualche volta, anche tramite una stampante. La *tastiera* (keyboard) è usata per inviare le informazioni al computer e, ogni volta che viene premuto un tasto, il codice elettronico corrispondente al carattere del tasto è inviato al computer, che lo riconosce ed agisce conseguentemente oppure lo ignora. La tastiera è il tuo dispositivo d'ingresso (detto anche *input*) che fornisce le informazioni al computer.

"Sono robusto, utile ed amichevole, quando mi si conosce bene".

Lo *schermo* o CRT (Tubo a Raggi Catodici) chiamato anche *display*, visualizza le informazioni generate dal programma. Generalmente, ogni carattere premuto sulla tastiera apparirà anche sullo schermo. Il carattere è, in un primo tempo, inviato al computer e, successivamente, "riflesso" sullo schermo. Infatti normalmente, non c'è una connessione diretta tra la tastiera e lo schermo, ma tutto passa attraverso il computer, come illustrato nella vignetta seguente.

"Questa è la mia tastiera".

"Ho bisogno di un tuo "input" per sapere cosa fare".

La tastiera invia informazioni al computer.

A seconda della struttura del sistema, il *computer* vero e proprio può essere assemblato in un contenitore separato o integrato insieme alla tastiera, allo schermo e/o alle unità a disco, ma indipendentemente dal contenitore, il computer è composto da un'unità di elaborazione, da una memoria e da una serie di interfacce (esse contengono le parti elettroniche per connettere stampanti o altri dispositivi). Esaminiamo, ora, questi tre elementi.

L'unità di elaborazione (detta comunemente CPU, cioè *Central Processing Unit*) prende le istruzioni dal programma, una dopo l'altra e le esegue. La CPU richiede pochi componenti che sono chiamati circuiti integrati o "chips". Tutti i microcomputers usano un chip *microprocessore* come elemento principale della CPU. Un chip tipico è mostrato nella vignetta qui sotto.

"Questo è un "chip" di microprocessore".

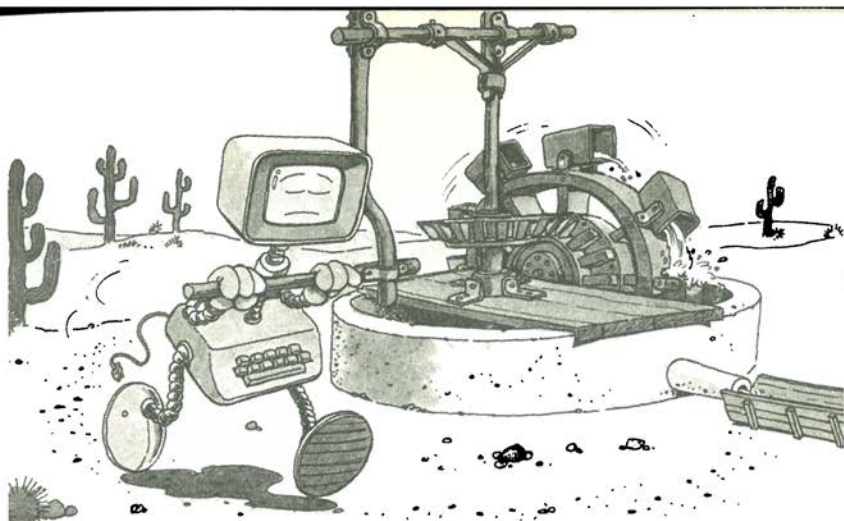
La *memoria* immagazzina i programmi e tutte le informazioni che i programmi manipolano, leggono o generano durante l'esecuzione. Per poter eseguire un programma, esso viene inizialmente posto nella memoria del computer. Per esempio, se il programma, originariamente, fosse memorizzato su una cassetta o un dischetto, dovrebbe essere prima trasferito nella memoria del computer. Questa operazione è detta caricamento del programma (in inglese *load*).
Nel computer deve esserci memoria sufficiente da potervi sistemare i programmi più grandi insieme con i dati che il programma manipolerà.
Nel tuo computer sono presenti due tipi di memoria: ROM e RAM. Il tipo "normale" di memoria che usi per caricare il tuo programma è la RAM (Random Access Memory cioè memoria ad accesso casuale); essa è una memoria di lettura e scrittura, cioè le informazioni possono essere scritte nella RAM e lette da essa. L'aspetto di una RAM somiglia al microprocessore mostrato qui sotto, ad eccezione che internamente c'è un chip diverso. Sfortunatamente, allo stato odierno della tecnologia, questo tipo di memoria è volatile, cioè il contenuto della RAM sparisce quando l'apparato viene spento.

Il tuo computer non farà niente, finchè non introdurrai un programma nella sua memoria.

Questo è il motivo per cui, alla fine di una sessione di lavoro, devi sempre memorizzare il tuo programma in un mezzo non-volatile, come una cassetta o un dischetto, se vorrai conservarlo. Ricordati, inoltre, che deve essere presente in memoria (o nella RAM o nella ROM) l'interprete BASIC, prima di eseguire il tuo programma BASIC.

ROM significa "Read Only Memory", cioè memoria di sola lettura. Questo tipo di memoria contiene permanentemente programmi caricati dal costruttore e che non possono essere cambiati. Essa è non-volatile e non si cancella mai.
Generalmente contiene un BASIC residente ed un programma speciale detto *monitor* che serve per comunicare col computer quando viene acceso.
Se la ROM non contiene nulla, il computer non saprebbe cosa fare quando premi i tasti sulla tastiera. Perciò, nella configurazione minima, la RAM deve contenere almeno il monitor. Il programma monitor esamina le informazioni inviate dalla tastiera e reagisce ad esse eseguendo azioni preordinate, come attivare l'interprete BASIC residente o caricare un programma da cassetta.

Non puoi usare la memoria ROM per memorizzare altri programmi, infatti tutti gli altri *programmi introdotti nel tuo computer vengono caricati nella RAM. Qualche volta, puoi* anche acquistare dei programmi su "cartridge" (cartuccia) per il tuo computer, in questo caso i programmi sono memorizzati su dei chips ROM dentro la "cartridge".
Almeno due dispositivi addizionali sono comunemente collegati ad un computer: una memoria di massa ed una stampante. Questi dispositivi vengono connessi attraverso le *interfacce* (cioè le parti elettroniche necessarie per il collegamento di dispositivi speciali). La memoria di massa può essere un registratore a cassette o una o più unità a dischi chiamate comunemente *disk drives.* (**Nota bene:** entrambi questi dispositivi usano un mezzo magnetico per registrare le informazioni e possono memorizzare molte più informazioni della memoria elettronica interna del computer). Questi dispositivi speciali richiedono una specifica interfaccia nel contenitore del computer per poter comunicare con lui.

La maggior parte dei personal computers hanno una interfaccia interna (coupling) per il registratore a nastro ed, invece, è richiesta una interfaccia con scheda separata per connettere una o più unità a dischi o una stampante.
Una stampante è necessaria per ottenere dei listati permanenti dei programmi e dei risultati. Una stampante, così come un display video, è un dispositivo di uscita (detto anche *output*) e sono previste delle istruzioni specifiche per mandare le informazioni o al video o alla stampante.

Per finire, un altro dispositivo spesso usato è il *modem*; esso permette di comunicare con un altro computer, o con un terminale, attraverso una normale linea telefonica. Esso è utile quando ci si allaccia ad una rete commerciale o per accedere alle banche di dati (raccolte di informazioni).
Abbiamo, a questo punto, imparato il vocabolario richiesto, però, prima di passare al capitolo 2 ed iniziare ad usare il computer, sono necessarie due parole di avvertimento.

Il programma "monitor" è sempre al lavoro, pronto ad eseguire tutti i lavori più umili.

Computer e sintassi

"Ricorda ... sii preciso".

I computers sono veloci, pazienti e precisi; essi fanno solo quello che gli si ordina di fare e lo fanno con esattezza. Per poter comunicare col tuo computer con successo, devi essere preciso. Se farai un errore o una imprecisione nello scrivere una istruzione BASIC, non danneggerai in alcun modo il tuo computer, ma il tuo programma non verrà eseguito correttamente e, generalmente, si fermerà, visualizzando "Syntax error" cioè errore di sintassi.
Ricordati che la *sintassi* è l'insieme di regole che specificano il modo corretto di scrivere una istruzione BASIC. Le regole di sintassi sono rigide. Per esempio, non puoi usare un'ortografia approssimata: se la regola specifica un punto, non puoi usare una virgola o un punto e virgola. Ogni carattere viene rigidamente interpretato dal computer ed ha un preciso significato. Ogni cambiamento significherebbe sbagliare o, almeno, ottenere dei risultati inaspettati.
Ricorda che se non obbedisci esattamente alle regole, il tuo programma probabilmente non funzionerà. Quando scrivi le istruzioni di un programma non cercare di essere creativo. Le regole sono semplici, chiare e facili da seguire. È meglio dirottare la tua creatività soprattutto nello studio del programma e nella pianificazione del lavoro che desideri portare a termine. In breve, è molto importante seguire attentamente le istruzioni e le raccomandazioni date in questo libro.

Capitolo **2**

Comunicare con il tuo computer

In questo capitolo, imparerai a comunicare con il tuo computer, cioè imparerai ad inviare al computer le istruzioni BASIC ed a visualizzare parole e frasi sul suo schermo. Le informazioni scambiate tra te ed il computer includeranno programmi (istruzioni in BASIC che tu invii) e dati (numeri e caratteri che tu invii o ricevi). Inizialmente imparerai ad usare la tastiera di un computer, così che potrai iniziare subito ad inviare istruzioni. In particolare, imparerai a muovere il cursore sullo schermo ed a correggere gli errori di scrittura. A questo punto invierai le tue prime istruzioni in BASIC e farai visualizzare alcuni messaggi sullo schermo del computer. Imparerai la differenza fra esecuzione *immediata* e *differita* ed alla fine, i passi necessari per scrivere un semplice programma e farlo funzionare.

Alla fine di questo capitolo, dovresti avere familiarizzato con i comandi essenziali per comunicare con il tuo computer.

L’uso della tastiera

Il disegno sottostante mostra una tastiera per computer.
Come puoi notare essa è uguale ad una normale tastiera per macchina da scrivere, con in più alcuni tasti speciali.

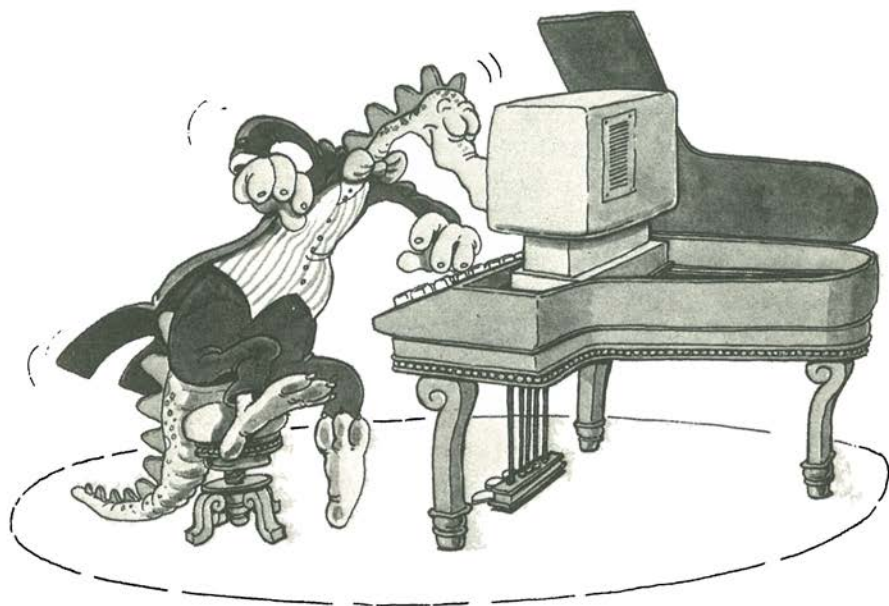
Una tastiera di computer.

Le tastiere per computer vengono prodotte in varie forme e combinazioni di tasti, ma, ad eccezione di pochi dettagli, hanno tutte le stesse caratteristiche. I tasti principali sono:

1. le lettere dell’alfabeto (dall’A alla Z)
2. i numeri da 0 a 9
3. alcuni simboli, come =, +, *, ” e $
4. un tasto chiamato “carriage return” (ritorno carrello), generalmente siglato “RETURN” o “CR” o ↵
5. un tasto chiamato SHIFT ed uno CTRL (controllo)

6. un tasto chiamato RUBOUT (cancella) ed uno chiamato ESC o BREAK (interruzione).

Alcune tastiere sono provviste anche di altre caratteristiche addizionali come un tastierino numerico separato (detto KEY PAD) ed altri tasti con funzioni speciali. Ora descriveremo l'impiego di tutti questi tasti.

Impara tutto sulla tastiera.

Caratteri, numeri e simboli

Lo scopo dei tasti delle *lettere*, dei *numeri* e dei *simboli* è evidente; essi vengono usati per gli stessi scopi dei tasti di una macchina da scrivere.

Carriage return (ritorno carrello)

Su una macchina da scrivere, il tasto o la leva del ritorno carrello esegue due azioni: riporta il carrello all'inizio della linea ed avanza la carta alla linea successiva. Il nome deriva da queste azioni. Sul computer, invece, il tasto RETURN generalmente porta il *cursore* all'inizio della riga seguente sullo schermo. (Il cursore è un quadratino lampeggiante o un trattino che indica la posizione dove verrà visualizzato il prossimo carattere sullo schermo).
Su un computer, il tasto RETURN potrebbe essere chiamato più appropriatamente il tasto d'INTRODUZIONE, infatti la sua funzione principale è quella di INTRODURRE nella memoria un carattere o una linea di testo o un dato. Per questo motivo, alcune tastiere, ora lo indicano con ENTER (introdurre). Altre lo indicano con ↵.
In ogni caso, ogni istruzione diretta al computer, incluse le istruzioni BASIC, devono finire con un RETURN, perchè il RETURN significa "introduci la linea in memoria".
Infatti, qualunque cosa sia stata scritta su una linea, viene ignorata finchè non si preme il tasto RETURN. In questo modo puoi correggere gli errori che hai fatto, prima di introdurre la linea nella memoria del computer.

Il RETURN viene usato solamente durante la scrittura da tastiera. Sebbene sia necessario, non viene memorizzato come parte della istruzione del programma. In questo capitolo introduttivo, per aiutarti ad imparare, mostreremo tutti i caratteri che devi scrivere sulla tastiera ed indicheremo il tasto RETURN con un simbolo alla fine di ogni linea. Esso verrà rappresentato con ↵

Ricordati: devi premere sempre il tasto RETURN per trasmettere un'istruzione al computer, altrimenti non accadrà nulla. Analogamente, quando il computer, in seguito, ti chiederà dei valori, dovrai, allo stesso modo, introdurre le risposte premendo RETURN.

Shift

Il tasto SHIFT lavora come il corrispondente tasto di una macchina da scrivere e ti permette di selezionare la scrittura in maiuscolo o in minuscolo o i simboli superiori o inferiori. Infatti, ad eccezione delle lettere, la maggior parte dei tasti hanno due simboli stampati sopra di essi, uno superiore ed uno inferiore. Finchè non spingerai il tasto SHIFT verranno generati, alla pressione del relativo tasto, i simboli inferiori. Quando premi il tasto SHIFT insieme con un altro tasto, verrà invece generato il simbolo superiore. Con i tasti delle lettere, normalmente si può scegliere tra scrittura in maiuscolo o in minuscolo.

Alcune tastiere semplificate, però, permettono la generazione dei soli caratteri maiuscoli, perchè i codici dei caratteri in minuscolo vengono utilizzati per altri simboli o funzioni. Il BASIC standard richiede che le istruzioni vengano scritte in maiuscolo, perciò in questo libro useremo nei nostri programmi solamente i caratteri maiuscoli.

Però se la tua tastiera possiede anche quelli minuscoli, puoi introdurre o manipolare i testi, usando anche questi ultimi.

La maggior parte delle tastiere possiedono anche il tasto CAPS LOCK che ti permette di bloccare la tastiera sul funzionamento in maiuscolo. Premendo di nuovo questo tasto si ritorna in minuscolo. Lo scopo del tasto SHIFT è di ridurre il numero totale dei tasti. Infatti, con la sua presenza, vengono raddoppiati il numero dei caratteri o dei comandi generati da ogni singolo tasto.

Facciamo, ora, un po' di pratica su ciò che hai imparato: vai alla tua tastiera e premi casualmente dei tasti. Premi pure qualunque tasto, perchè non danneggerai nulla. Usa il tasto SHIFT ed osserva quali caratteri vengono visualizzati sullo schermo. Ora premi RETURN e vedi cosa accade.

Tasto di controllo (CTRL)

Il tasto CONTROL (o CRTL), che non esiste sulle macchine da scrivere, serve per introdurre velocemente dei comandi al computer. Il tasto CONTROL si usa come il tasto SHIFT: premendolo e mantenendolo premuto mentre si preme un altro tasto. Questo procedimento è detto generazione di un carattere di controllo. Per esempio, "CTRL A" viene generato premendo contemporaneamente il tasto CTRL ed A. Questo è un sistema utile per introdurre un comando o un codice di controllo, premendo solamente due tasti. (**Nota bene:** i *codici di controllo* sono previsti per facilitare l'uso dei comandi più frequentemente usati, come il movimento del cursore (a destra, a sinistra, in alto, in basso, di una posizione, di una parola, di una frase) e per cancellare o inserire un carattere o una intera linea di testo. L'uso di questi caratteri è specifico di ogni programma usato sul tuo computer ed il loro effetto è spiegato nella documentazione del relativo programma. In questo libro non ne useremo nessuno, ma puoi, comunque, cercare di impararne almeno uno: il codice per interrompere un programma sbagliato. (Spesso è CTRL C)

RUBOUT

Il tasto RUBOUT o "tasto di cancellazione" è usato per evidenti motivi, ma può non essere incluso su quelle tastiere che compiono la stessa funzione (cancellazione di un carattere) posizionando il cursore un passo indietro e scrivendo sopra al carattere precedente.

ESC o BREAK

Il tasto ESC o BREAK è fornito come carattere di controllo "standard" comune a tutti i programmi; esso ti permette, premendolo, di arrestare ogni elaborazione.
Qualunque programma, sia l'interprete BASIC che un tuo programma, deve possedere un comando speciale, come END o EXIT, che ne concluda l'esecuzione. Comunque, in

quei casi ove stia accadendo qualcosa di inaspettato e tu voglia interrompere immediatamente l'esecuzione del programma, puoi generalmente usare il tasto ESC. Se, come spesso accade, puoi usare anche un codice di controllo (come CTRL C), allora usa preferibilmente il tasto CTRL per compiere questa operazione.

Il tastierino numerico

Un tastierino numerico.

Il tastierino numerico, detto anche KEY PAD, (mostrato a sinistra) è molto utile in applicazioni di lavoro, quando si debbano introdurre rapidamente molti numeri. Il tastierino numerico è utilizzato nella serie di tastiere per computer da scrivania.
I suoi tasti, semplicemente, duplicano le funzioni dei tasti normali della tastiera regolare (da 0 a 9, +, —, x, ÷ e =).

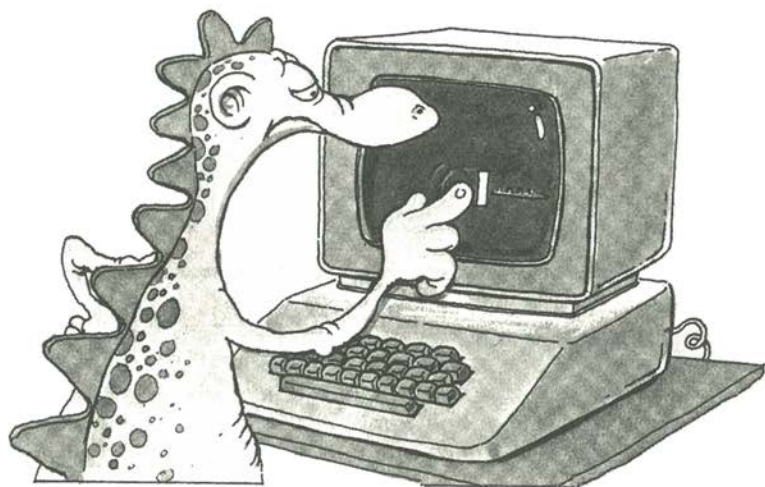
I tasti di funzione

I tasti di funzione sono una caratteristica utile per inviare al computer i comandi, più frequentemente usati.
Infatti, premendo un singolo tasto si ha lo stesso effetto che scrivere un lungo comando come PRINT (stampa) o EXECUTE (esegui).
Perciò puoi premere un solo tasto per cancellare un carattere, inserirne una sequenza o eseguire un programma.
Questi tasti sono specifici per ogni ditta costruttrice e risparmiano molto lavoro di scrittura. I tasti più utili sono: scroll (tutto il testo sale di una riga), clear (cancellazione di tutto lo schermo), cambiamento di pagina (avanti o indietro) e i tasti di posizionamento del cursore: ↑, ↓, ←, → e home (riporta il cursore nell'angolo in alto a sinistra).

Il cursore

Ricordati che il cursore è un quadratino o una lineetta che indica la tua posizione corrente sullo schermo.
Generalmente lampeggia in modo che puoi vederlo facilmente.
Ecco un cursore che ti sta mostrando dove sei sullo schermo dopo aver scritto SALVE:

```
SALVE □
```

Muovendo il cursore avanti e indietro, su e giù sullo schermo, puoi scrivere sopra i caratteri e modificare qualunque testo che hai scritto precedentemente. Questo è un grosso vantaggio per effettuare variazioni e potrai usare il cursore estesamente per correggere errori di scrittura.
La maggior parte delle tastiere sono fornite di almeno quattro tasti di funzione per posizionare il cursore sullo schermo: ↑, ↓, ←, →.

Impara a muovere il cursore sullo schermo.

Desidererai ora tornare alla tua tastiera per far pratica delle cose appena imparate. Quando avrai preso una ragionevole familiarità con la tua tastiera, ritorna al libro per imparare come introdurre le istruzioni in BASIC.

Parlare in BASIC

Per parlare in BASIC con il tuo computer, hai bisogno di un interprete BASIC nella memoria. Se hai un interprete BASIC residente, generalmente non devi fare niente, anche se alcuni sistemi richiedono che tu batta:

B ↵

oppure

BASIC ↵

per attivare l'interprete. Se il tuo computer, invece, non ha un BASIC residente, devi caricarlo dal dischetto o dalla cassetta.
Quando l'interprete è attivato, riceverai una conferma del tipo:

```
XBASIC 2.1 READY
>
```

dove XBASIC è il nome dell'interprete, 2.1 è la versione e READY significa "pronto ad operare". Poichè saranno disponibili continuamente nuove versioni, questo numero serve a distinguerle una dall'altra. > è il *prompt*. (Il simbolo del prompt può variare, ma in questo libro useremo sempre il simbolo >). Il prompt è un messaggio dell'interprete BASIC che significa "Sono pronto. Incomincia a dirmi cosa devo fare." Non appena vedrai questo simbolo, saprai che l'interprete BASIC è pronto ed aspetta le tue istruzioni. Andremo avanti assumendo che:

1. Un interprete BASIC sia nella memoria del tuo computer.
2. Il prompt del BASIC usato dal tuo interprete (identificato dal simbolo > appaia sullo schermo. Se non dovesse apparire, premi RETURN e guarda cosa succede. Se ciò non basta, spegni il tuo computer e riaccendilo.

Invieremo ora al computer la nostra prima istruzione BASIC. Scrivi quanto segue (esattamente come appare qui):

```
XBASIC 2.1 READY
>PRINT "SALVE" □
```

Lo schermo apparirà così (ricordati che "PRINT in inglese significa "stampa"):

Il > sulla sinistra è il *prompt* che ti sta avvertendo che l'interprete sta aspettando una istruzione ed i caratteri sulla destra sono quelli che hai appena scritto, ma non succede niente. Ti ricordi il perchè?
Ciò accade perchè devi premere il tasto RETURN per *introdurre* la tua istruzione. Ora premi RETURN e il tuo schermo mostrerà:

```
XBASIC 2.1 READY
>PRINT "SALVE"
SALVE
>□
```

L'interprete ha ricevuto la tua istruzione e l'ha eseguita immediatamente visualizzando SALVE come tu hai richiesto. Poi l'interprete ha visualizzato un nuovo prompt (>), avvertendoti di essere pronto per una nuova istruzione.
Esaminiamo ora, più dettagliatamente, la nostra prima istruzione BASIC:

PRINT "SALVE"

Questa istruzione è composta da due parti: PRINT e "SALVE". PRINT è una *parola riservata* che ha uno specifico significato per l'interprete BASIC. "SALVE" è il messaggio da visualizzare. Si può usare qualsiasi messaggio, ma deve essere racchiuso tra le virgolette.
Proviamone un'altro. Scrivi:

PRINT "QUESTA È UN'ALTRA PROVA" ↵

Sul tuo schermo apparirà quanto segue:

```
>PRINT "QUESTA E' UN'ALTRA PROVA"
QUESTA E' UN'ALTRA PROVA
>□
```

Ora prova di nuovo, visualizzando un tuo messaggio, ma ricorda che se dimentichi una delle due virgolette o scrivi male PRINT, l'istruzione non verrà eseguita e ti verrà inviato un messaggio d'errore. Ora, rompi gli indugi e prova: non danneggerai nulla.
Potrai chiederti perchè questa istruzione si chiami PRINT che significa "stampa" quando l'azione è una semplice visualizzazione sullo schermo e non una stampa. Anche se tu avessi una stampante collegata al tuo computer, non verrebbe stampato nulla. Forse avrai già indovinato la risposta. I primi terminali erano dei dispositivi simili alle macchine da scrivere che, perciò, stampavano (e non visualizzavano) le informazioni e sebbene la tecnologia sia cambiata, le istruzioni sono rimaste le stesse. Per inviare le informazioni alla stampante, piuttosto che al video, bisogna usare un'altra istruzione BASIC chiamata LPRINT. Il suo nome deriva dal fatto che il suo scopo principale è di *LISTare* i programmi sulla stampante.
Prima di andare avanti, assicurati che il prompt sia apparso sullo schermo; cioè che l'interprete BASIC sia pronto a ricevere un'altra informazione, perchè in caso contrario, non potresti eseguire nessun comando BASIC. In tal caso, premi RETURN o CTRL C, o, se ancora non funziona, spegni e riaccendi il tuo sistema.
Ora, piuttosto che eseguire una singola istruzione (statement), scriveremo il nostro primo programma in BASIC.
Scrivi quando segue:

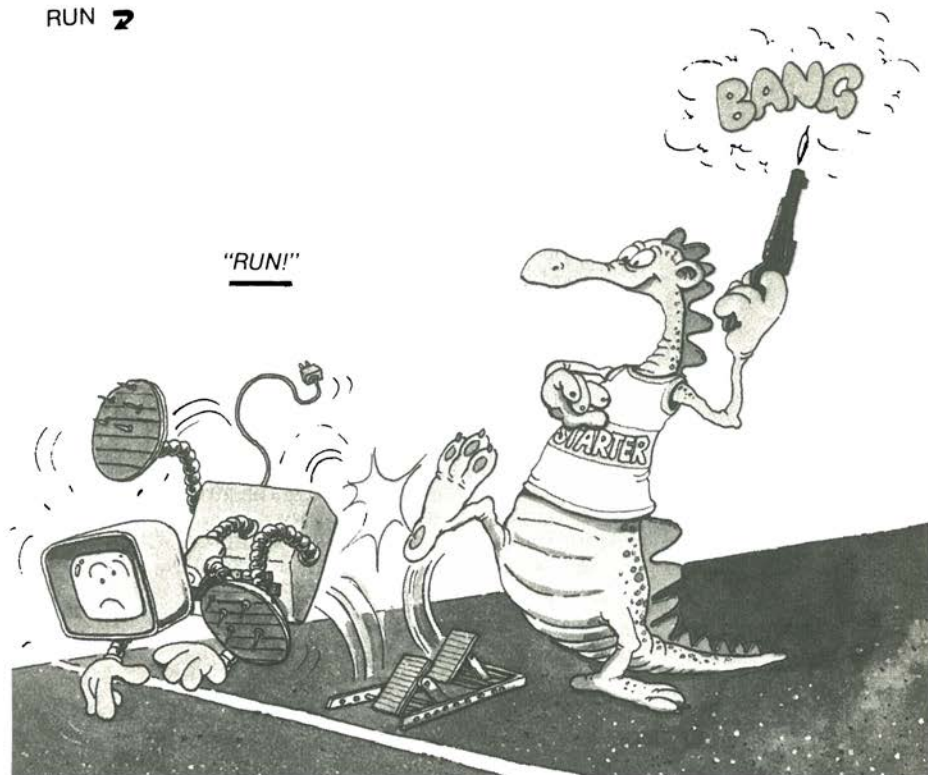
```
10 PRINT "SALVE" ↵
20 PRINT "COME STAI ?" ↵
30 END ↵
```

Il tuo schermo apparirà così:

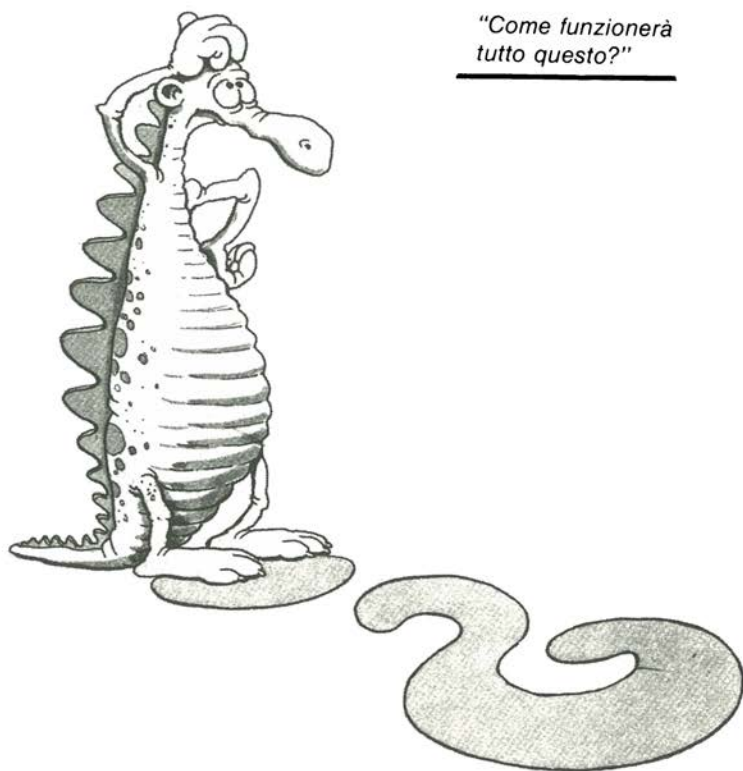
```
>10 PRINT "SALVE"
>20 PRINT "COME STAI ?"
>30 END
>□
```

Sarai sorpreso che non accadeva nulla e che il computer rispondeva semplicemente con un >. Queste tre linee sono più che tre corrette istruzioni BASIC, esse costituiscono un breve *programma BASIC.* Nota che ogni linea inizia con un numero che viene detto numero di linea o *etichetta.* Essa dice al computer che quanto andiamo a scrivere è un programma e non una istruzione da eseguire immediatamente. Ora scrivi:

RUN ↵

"RUN!"

(**Nota bene**: questo comando (RUN significa anche "esegui") può variare da un interprete all'altro).

Ora dovrai vedere sul tuo schermo:

"Come funzionerà tutto questo?"

o, secondo il tuo interprete:

Come funziona tutto questo? Noi creiamo, inizialmente, un programma di tre linee e lo memorizziamo, una linea alla volta, premendo RETURN. Poi lo eseguiamo scrivendo RUN.
Questo è il metodo tipico per scrivere un programma e poi eseguirlo. D'ora in avanti seguiremo sempre questa procedura. Tutte le linee devono essere precedute da una etichetta numerica e il programma le eseguirà nell'ordine delle etichette.
Ogni volta che in risposta ad un >, scriverai una istruzione BASIC senza etichetta, essa verrà eseguita immediatamente e non memorizzata. Questo è chiamato *modo immediato* o *modo di calcolo*. Ogni volta, invece, che scriverai, in risposta ad un >, una istruzione BASIC con l'etichetta, essa verrà memorizzata, ma non eseguita finchè non scriverai RUN. Questo è chiamato *modo differito* o normale. Ora dimostriamo come funziona.
Il nostro programma di tre linee è stato memorizzato e potrà essere eseguito quante volte vorremo oppure potrà essere ingrandito o modificato. Ora scrivi:

RUN

e, di nuovo, vedrai:

Esaminiamo la memoria del computer visualizzandone il contenuto. Per far questo scrivi:

LIST

e, sul tuo schermo, dovresti vedere quanto segue:

```
10 PRINT "SALVE"
20 PRINT "COME STAI ?"
30 END
>□
```

Il tuo programma è LISTato sul video come è immagazzinato nella memoria.
Se hai una cassetta o un dischetto, puoi memorizzare il tuo primo programma e, per far questo, il comando, normalmente, è:

SAVE ↵

Potrai, in seguito, riprenderlo con il comando:

LOAD ↵

Per dimostrare la differenza tra istruzione immediata e differita, scrivi:

PRINT "CIAO" ↵

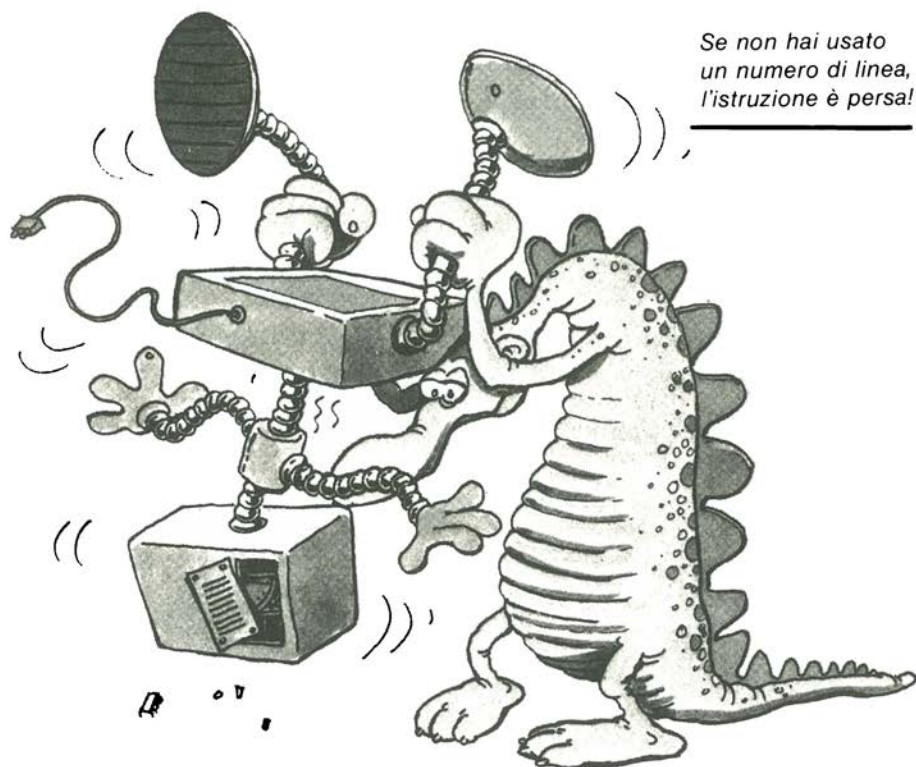
e sul tuo schermo vedrai:

Ora scrivi:

LIST ↵

e, di nuovo, vedrai sul tuo schermo:

```
10 PRINT "SALVE"
20 PRINT "COME STAI ?"
30 END
>□
```

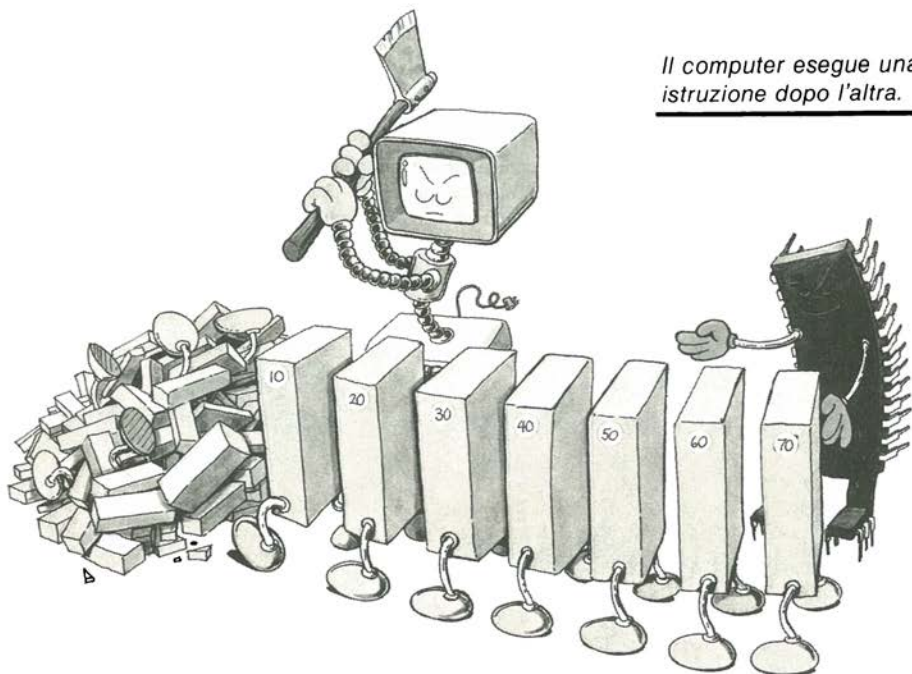
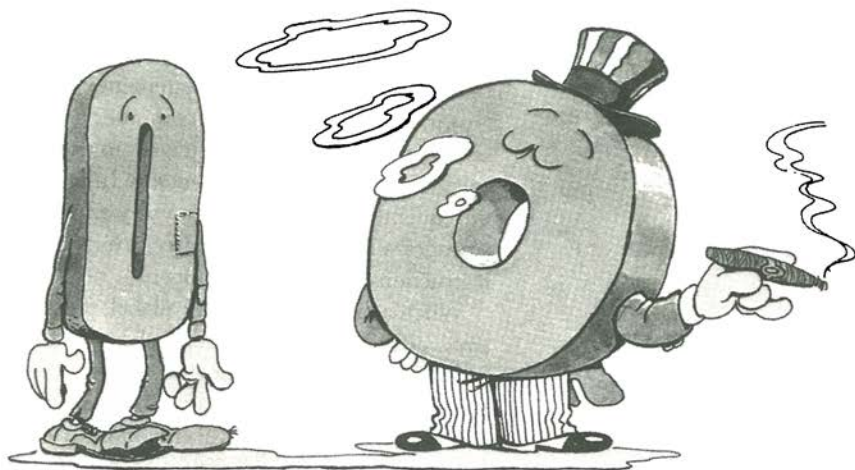
La nuova istruzione, PRINT "CIAO", non c'è più: non è stata memorizzata.

Ricapitolazione sui programmi

Per concludere, un'istruzione immediata viene eseguita non appena la scrivi, non viene memorizzata nella memoria del computer e dovrai riscriverla ogni volta che vorrai eseguirla. Questa possibilità è chiamata modo ad *esecuzione immediata*. In pratica,

questo metodo è usato poco frequentemente: generalmente quando vuoi verificare alcuni valori dopo che un programma è stato arrestato. Esperimenta liberamente alcune istruzioni BASIC ed osserva cosa accade.
Quando una linea è preceduta da una etichetta, il modo è chiamato ad *esecuzione differita*. In questo modo, ogni linea è memorizzata nella memoria del computer e, quando è stato introdotto interamente il programma, lo si esegue con il comando RUN. Ricordati che quando spegni il computer, il contenuto della RAM svanisce, incluso il programma che hai scritto. Perciò se vuoi conservare il programma per un uso futuro, devi salvarlo (SAVE) su di una cassetta o su di un dischetto.
Ogni linea di un programma BASIC deve iniziare con una etichetta che specifica l'ordine in cui la linea verrà eseguita. Perciò la linea 10 verrà eseguita prima della linea 20.
Il comando END, la linea 30 del nostro esempio, è opzionale in molti BASIC moderni, mentre è obbligatorio in quelli più vecchi. L'istruzione END dice all'interprete, durante l'esecuzione, che ha raggiunto una fine del programma legittima e non accidentale.

Come dettaglio finale, osserva che il numero 0 deve apparire diverso dalla lettera O; per evitare confusione il numero 0 viene rappresentato con Ø.
Comunque, oggi, uno zero può essere facilmente visualizzato sullo schermo con un simbolo più sottile della O.

Il computer esegue una istruzione dopo l'altra.

Non confondere la lettera O con il numero 0.

Un programma più lungo

RUN, LIST e SAVE sono chiamati *comandi*. Esse sono parole riservate che vengono usate da sole per specificare azioni speciali eseguite dal sistema, o più precisamente dall'interprete. Anche questi comandi fanno parte del BASIC. Progressivamente, impareremo molte altre istruzioni e comandi andando avanti con questo libro.
Scriviamo un nuovo programma più completo, usando l'istruzione PRINT ed il comando NEW (nuovo). Scrivi quanto segue:

NEW ↵

e, poi:

LIST ↵

Non accade niente: non c'è più alcun programma nella memoria, infatti il comando NEW ha cancellato la memoria del computer. Ora scrivi:

```
NEW

10 PRINT "QUESTO" ↵
20 PRINT "E'" ↵
30 PRINT "UN ALTRO" ↵
40 PRINT "ESEMPIO" ↵
50 END ↵
```

Vediamo ora cosa accade. Scrivi:

```
RUN ↵
```

e, ora,.sullo schermo dovresti vedere

```
QUESTO
E'
UN ALTRO
ESEMPIO
>□
```

A seconda del tuo BASIC, può apparire o no un "END" alla fine del programma. Il nostro programma visualizza esattamente, il testo atteso. Verifichiamo che il programma sia stato memorizzato correttamente, scrivendo:

```
LIST ↵
```

E ora sul tuo schermo dovresti vedere:

```
>LIST
10 PRINT "QUESTO"
20 PRINT "E'"
30 PRINT "UN ALTRO"
40 PRINT "ESEMPIO"
50 END
>□
```

Il comando NEW è stato usato per pulire la memoria, cioè per cancellarla e fare spazio per un "nuovo" programma. Se non avessimo usato il comando NEW, le nuove istruzioni avrebbero cancellato e rimpiazzato ogni vecchia istruzione con identica etichetta scritta precedentemente in memoria. Questo potrebbe portare ad errori se una istruzione residua rimanesse in un programma nuovo. Se non usi il NEW, ogni volta che scriverai una istruzione con l'etichetta 10, la nuova versione cancellerà automaticamente ogni versione precedente dell'istruzione 10. Ma, *attento*, se prima avevi scritto una istruzione con l'etichetta 15, poi nel programma scritto senza un NEW, la istruzione "15" rimarrà nella memoria del computer e sarà automaticamente incorporata nel nuovo programma al suo posto (poichè l'etichetta 15 non è usata nel nuovo programma). Ora dimostreremo quanto detto. scrivi:

15 PRINT "* * * * *" ↵

e, poi

RUN ↵

Il tuo schermo mostrerà questo:

```
QUESTO
* * * * *
E'
UN ALTRO
ESEMPIO
>□
```

Come puoi vedere, la nuova istruzione PRINT, con l'etichetta 15, è stata automaticamente inserita nel vecchio programma dopo l'istruzione 10. Listiamo il programma. Scrivi:

LIST ↵

Il tuo schermo mostra quanto segue:

```
10 PRINT "QUESTO"
15 PRINT "* * * * *"
20 PRINT "E'"
30 PRINT "UN ALTRO"
40 PRINT "ESEMPIO"
50 END
```

Puoi verificare che l'istruzione 15 fa parte, ora, del tuo programma. Ricordati che ogni volta che scrivi una istruzione con una etichetta, essa è inserita automaticamente, al suo posto, nella sequenza del programma.
Dimostreremo ora che quando usi un'etichetta che *già esiste*, la nuova istruzione cancellerà automaticamente quella precedente. Usiamo questa proprietà per cancellare l'istruzione 15. Scrivi:

15 PRINT "........" ⏎

poi:

RUN ⏎

ed il tuo schermo mostrerà:

```
QUESTO
........
E'
UN ALTRO
ESEMPIO
>□
```

Come puoi vedere, la tua nuova istruzione PRINT con l'etichetta 15 si è sovrapposta a quella precedente.
Verificalo con il comando:

LIST ⏎

e sullo schermo vedrai:

```
10 PRINT "QUESTO"
15 PRINT "........"
20 PRINT "E'"
30 PRINT "UN ALTRO"
40 PRINT "ESEMPIO"
50 END
>□
```

Per evitare errori, quando scrivi un nuovo programma, dovresti usare l'istruzione NEW per cancellare la memoria del computer ed evitare interferenze causate da istruzioni "residue" dei programmi precedenti.

"Io cancello così!".

Cancelliamo, ora, l'istruzione 15. Ci sono molti metodi per farlo. Scriviamo:

15 ↵

Questa istruzione è composta solamente dall'etichetta e si chiama *istruzione vuota.* Infatti, l'istruzione 15 non fa nulla, eccetto che cancellare ogni precedente versione. Ora scrivi RUN e, sullo schermo, dovresti osservare quanto segue:

```
QUESTO
E'
UN ALTRO
ESEMPIO
>□
```

Stai attento, perchè questo metodo può essere pericoloso. Infatti se, per errore, scrivi

20

e premi RETURN, cancelli la precedente versione dell'istruzione 20 e la rimpiazzi con una "vuota" che non esegue nulla. Per evitare sorprese, verifica sempre il listato del tuo programma prima di eseguirlo.

Riassunto

Abbiamo imparato a scrivere dei semplici programmi BASIC che visualizzano informazioni sullo schermo, abbiamo scritto un programma BASIC usando le istruzioni etichettate ed abbiamo discusso perchè i programmi dovrebbero essere preceduti da un comando NEW e terminare con un END. Abbiamo anche visto che un programma viene memorizzato automaticamente quando viene introdotto e può essere eseguito con il comando RUN.

Inoltre abbiamo visto come si può visualizzare il listato di un programma con il comando LIST e per finire, abbiamo usato il comando SAVE per salvare un programma su nastro o su disco.
Abbiamo imparato che le istruzioni di un programma vengono eseguite nell'ordine delle etichette e se duplichi un numero di etichetta intenzionalmente o accidentalmente, la nuova istruzione cancellerà automaticamente ogni istruzione precedente con la stessa etichetta. Inoltre, ogni volta che aggiungi una linea con una nuova etichetta, l'interprete la inserirà automaticamente al suo posto nella sequenza del programma.

In questo capitolo, abbiamo introdotto molti concetti nuovi e, se desideri veramente imparare a programmare, è essenziale che tu cominci a fare molta pratica su ciò che hai imparato. Seguono alcuni esercizi che sei fortemente consigliato di risolvere. Le risposte ad alcuni di essi sono alla fine del libro.

2.1: Scrivi un programma che visualizzi:

TI AUGURO UNA BUONA GIORNATA"

2.2: Scrivi un programma che visualizzi:

AAAAA

BBBB

CCC

DD

E

2.3: Scrivi un programma che visualizzi:

```
***********
  TITOLO
***********
```

2.4: Definisci i seguenti termini:

a. etichetta

b. esecuzione differita

c. esecuzione immediata

d. istruzione vuota

e. cursore

f. tasto di controllo

g. key pad o tastierino numerico

h. parola riservata
i. prompt

2.5: Quale è la differenza tra PRINT e LPRINT?

2.6: Puoi eseguire un intero programma scrivendo le istruzioni una alla volta nel modo immediato?

2.7: Perchè si usa NEW prima di scrivere un nuovo programma?

2.8: Puoi introdurre delle istruzioni di programma (con etichetta) fuori della sequenza?

2.9: Scrivi degli esempi di comandi BASIC.

2.10: La seguente istruzione è un modo valido per visualizzare la parola ESEMPIO?

PRINT ESEMPIO

2.11: Perchè si usa il tasto RETURN?

2.12: Spiega come si può cancellare l'istruzione 20 in un programma.

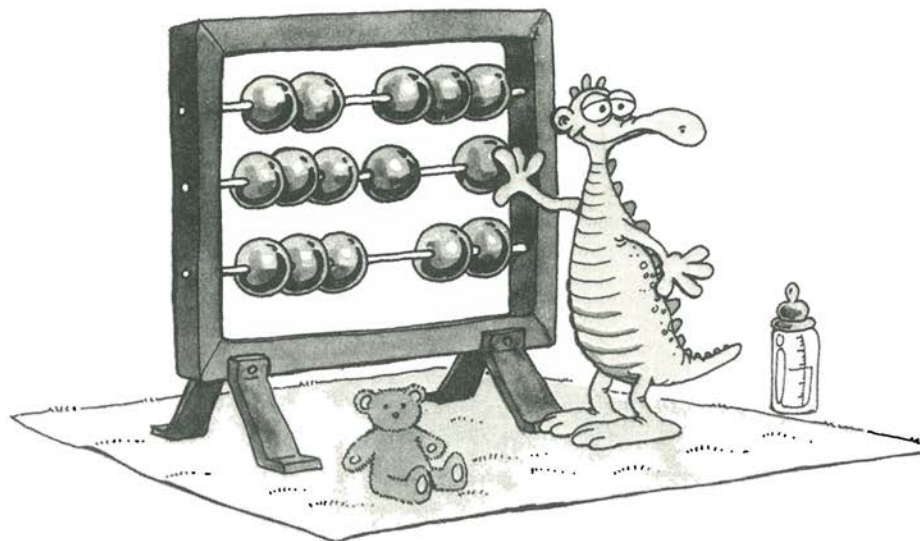
2.13: Se hai già scritto precedentemente l'istruzione 30 e desideri sostituirla con una nuova istruzione 30, devi prima cancellare quella vecchia o no?

2.14: Scrivi un programma che visualizzi quanto segue:

```
TTTTTT   H   H   EEEEEE
  TT     H   H   EE
  TT     H   H   EE
  TT     HHHH    EEEE
  TT     H   H   EE
  TT     H   H   EE
  TT     H   H   EEEEEE
```

Capitolo **3**

Far calcoli con il BASIC

In questo capitolo, inizieremo ad usare i numeri: li visualizzeremo, ci eseguiremo addizioni, sottrazioni, moltiplicazioni e divisioni. Impareremo ad eseguire i calcoli usando gli operatori aritmetici e descriveremo gli altri importanti operatori propri del BASIC.

Visualizzare i numeri

Fino ad ora abbiamo visualizzato solo i testi, proviamo ora con i numeri. Scrivi:

PRINT 3 ↵

Il risultato dovrebbe essere:

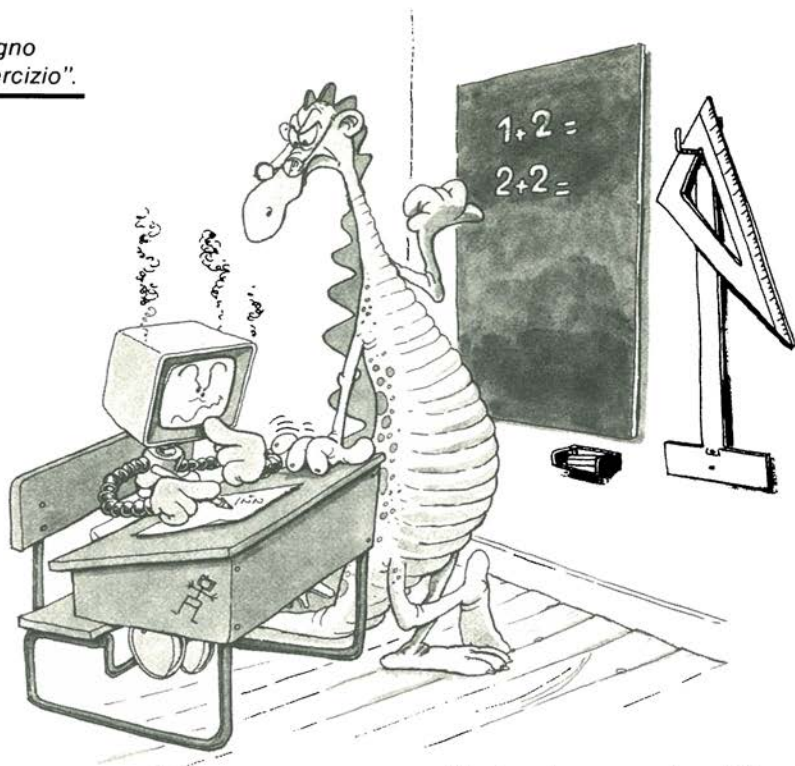
```
3
```

Ricordati che, per quanto detto nel Capitolo 2, questa istruzione è nel modo immediato perchè non è preceduta da una etichetta e viene eseguita immediatamente. In questo capitolo, scriveremo tutti gli esempi nel modo immediato, così potrai eseguirli premendo pochi tasti.
Nota che nel BASIC i numeri non vanno racchiusi tra le virgolette come i testi. L'uso delle virgolette permette di far distinguere all'interprete il testo inviato dall'utente dalle parole riservate, come PRINT. Il testo tra le virgolette è chiamato *stringa* e può contenere anche i numeri.
Proviamo ora a visualizzare dei numeri sempre più grandi, come 100, 1000, 10000, etc. Ad un certo punto, il tuo interprete BASIC si rifiuterà di visualizzare il numero ed invierà un messaggio del tipo "NUMBER TOO LARGE" (numero troppo grande). Ogni interprete BASIC impone un limite al massimo numero intero che può trattare. Se tu avessi bisogno di trattare dei numeri più grandi del limite imposto dal tuo interprete, dovresti generalmente usare un interprete diverso che ti permetta di usare valori più grandi.

Ricordati che un interprete BASIC che permette l'uso dei numeri interi è chiamato Integer BASIC. Un Integer BASIC non permette, però, l'uso dei decimali. Proviamo a vedere se il tuo interprete può trattare i decimali. Scrivi:

PRINT 1,5 ↵

Se il tuo schermo visualizzerà 1.5, allora il tuo BASIC può trattare i decimali; in caso contrario, riceverai probabilmente un messaggio d'errore o apparirà un "?".
Nel gergo dei computer, i numeri decimali sono chiamati *floating-point* cioè numeri a virgola mobile. Un interprete BASIC che permetta l'uso dei decimali è detto *floating-point BASIC.*

"Hai bisogno di più esercizio".

Notazione scientifica

Parliamo ancora dei numeri decimali. Come per i numeri interi, nel caso dei numeri decimali, l'interprete conserva solamente un certo numero di cifre. Per esempio il corretto valore di un terzo è:

0,3333333333...(etc.)

Nel computer, questo valore potrà essere memorizzato come:

0,3333333 (sono conservate le prime sette cifre dopo lo zero)

Si dice, perciò, che il valore esatto è stato *troncato* alla settima cifra. (**Nota bene**: questa è un'approssimazione, ma è generalmente sufficiente).
Se il tuo interprete BASIC permette l'uso dei decimali, allora, permette anche l'uso della *rappresentazione scientifica* per questi numeri. Quando un numero diviene o troppo piccolo o troppo grande, esso verrà visualizzato in notazione scientifica per guadagnare spazio.

Ecco un esempio:

3,2E6

che significa:

$$3{,}2 \times 10^6 = 3.200.000$$

10^6 significa 10 elevato alla sesta potenza, cioè 10 moltiplicato per 6 volte se stesso:

$$10 \times 10 \times 10 \times 10 \times 10 \times 10 = 1.000.000$$

Allo stesso modo:

1,12E — 7

significa:

$$1{,}12 \times 10^{-7} = 0{,}0000000112$$

10^{-7} significa 1/10 alla settima potenza, cioè 1/10 moltiplicato sette volte per se stesso (1/10 equivale a 10^{-1}).

Fare dell'aritmetica

Eseguiamo ora alcuni calcoli aritmetici. Scrivi:

```
PRINT 2 + 2
```

Il risultato che apparirà sul tuo schermo dovrebbe essere:

Abbiamo eseguito la nostra prima operazione aritmetica.
Il simbolo di addizione, +, è chiamato *operatore*.
Un operatore è un simbolo che rappresenta l'operazione che deve essere eseguita su di uno o più operandi. Il BASIC ha almeno cinque operatori aritmetici:

—	(meno)
+	(più)
*	(moltiplicazione)
/	(divisione)
ˆ o **	(elemento a potenza) (ˆ talvolta è indicato con ↑ [ )

Prova ora questo esempio. Scrivi:

PRINT 2 * 3

Il risultato dovrebbe essere 6. Il simbolo * è il simbolo della moltiplicazione perchè quello usuale, x, potrebbe essere confuso con la lettera X, per questo motivo tutti i linguaggi programmativi usano al suo posto il simbolo *.

Ecco altri esempi di valide istruzioni aritmetiche:

PRINT 1+2*3

Il risultato è

7

PRINT 3 — 2

Il risultato è

1

PRINT 8 / 2

Il risultato è

4

PRINT 1+2+3+4

Il risultato è

10

Se il tuo interprete BASIC permette l'uso dei numeri decimali, anche la seguente istruzione è esatta:

PRINT (6/3+12/4)/2

Il risultato è

2.5

Nota che, in questo esempio, sono state usate le parentesi per rendere più chiaro il raggruppamento delle operazioni. Scrivi:

PRINT 2+3+4/2

Il risultato è

7

La divisione (/) è stata eseguita per prima sul numero 4. Questo è dovuto al fatto che nel BASIC se lasciate a scelta, cioè senza usare le parentesi, la divisione (/) e la moltiplicazione (*) vengono eseguite prima delle addizioni (+) e delle sottrazioni (—). Se tu avessi l'intenzione di dividere il gruppo 2+3+4 per 2, sarebbe necessario scrivere:

PRINT ((2+3+4)/2)

Il risultato sarebbe:

4.5

Effettivamente la divisione è stata eseguita sul gruppo (2+3+4). È un buon metodo, per evitare confusione, quello di usare senza parsimonia le parentesi. Per esempio, la seguente espressione (o gruppo di valori ed operatori).

$$\frac{1+2+3}{4+5} \times 3$$

potrebbe essere tradotta nella seguente espressione BASIC:

((1+2+3)/(4+5))*3

oppure

(1+2+3)/(4+5)*3

perchè l'esecuzione procede da sinistra a destra quando gli operatori hanno la stessa precedenza, cioè la divisione viene eseguita prima della moltiplicazione.
Se tu scrivessi in BASIC quanto segue:

(1+2+3)/(4+5)*3

sarebbe equivalente a:

$$\frac{1+2+3}{(4+5)\times 3}$$

perciò usa le parentesi per separare i gruppi e assicurati che ogni parentesi sinistra abbia la corrispondente destra. Impariamo alcuni nuovi "trucchi" per visualizzare i valori.

Formati di stampa

Se scrivi:

PRINT "IL PRODOTTO DI DUE PER TRE È", 2*3 ⏎

Il risultato sarà:

```
IL PRODOTTO DI DUE PER TRE E'  6
```

Nella precedente istruzione PRINT, abbiamo unito, separandoli con una virgola, testo e numeri. Più precisamente abbiamo usato una *espressione*, 2*3, piuttosto che un numero. Ora scrivi:

ed avremo:

```
IL PRODOTTO DI DUE PER TRE, 2*3
```

Anche quest'ultima è una istruzione BASIC esatta, però non è quella che intendevi eseguire. Ricordati che ogni cosa scritta tra le virgolette viene visualizzata letteralmente. La virgola o il punto e virgola devono stare *fuori dalle virgolette* per funzionare correttamente.
Una istruzione PRINT può essere usata per visualizzare più informazioni (numeri o messaggi) sulla stessa linea.
Ogni informazione, però, deve essere separata da una virgola o un punto e virgola. Un punto e virgola metterà tra le singole voci un piccolo spazio, mentre la virgola ne metterà uno più grande. Come il tabulatore della macchina da scrivere, la virgola è usata per creare le *tabulazioni*, cioè dei campi sullo schermo. Questa tecnica è utile per visualizzare le tabelle.
Proviamo questa nuova funzione. Scrivi:

PRINT 1;2;3 ⮐

Il tuo schermo mostrerà:

```
1 2 3
```

Ora scrivi:

PRINT 1,2,3 ⮐

ed il tuo display ora mostrerà:

```
1    2    3
```

Calcoliamo il valore dell'IVA per una vendita di 1234 Lire. La percentuale dell'IVA è il 6.5%. Per far questo, presumeremo che il tuo interprete accetti i numeri decimali.
L'istruzione è:

PRINT "L'IVA È; 1234*6.5/100 ⮐

Il risultato è:

```
L'IVA E' 80.21
```

Potevamo anche scrivere:

```
PRINT "L'IVA È"; 1234*0,65
```

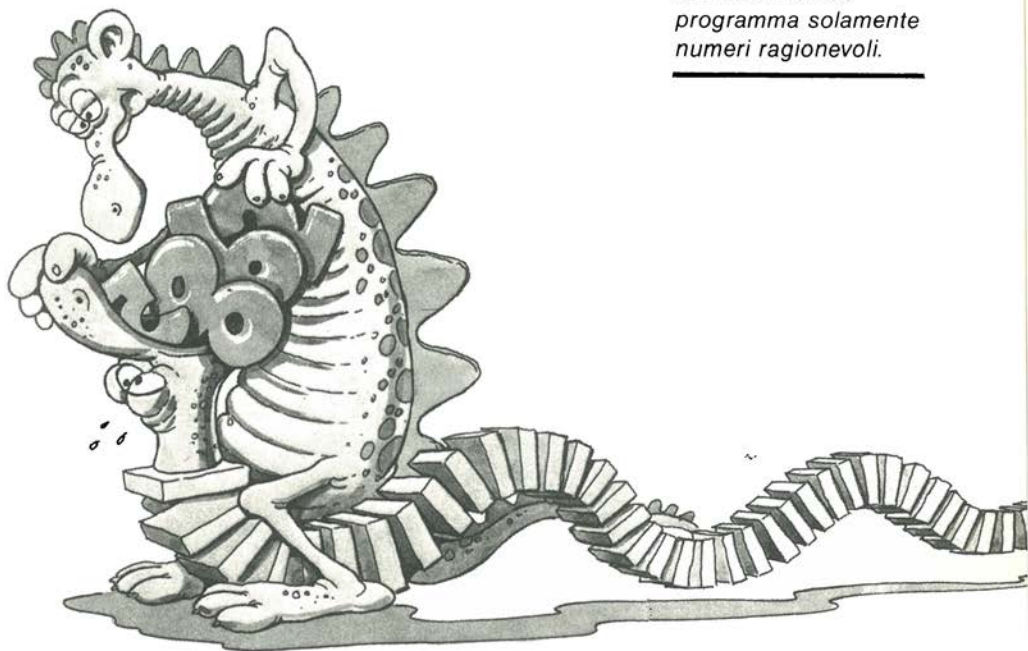
ed avremmo ottenuto lo stesso risultato. Ci sono perciò molti metodi equivalenti per scrivere un programma.
Puoi scrivere molte voci sulla stessa linea. Guarda:

```
PRINT 1;2;3;4;5;6;7;8;9; "MOLTE VOCI"
```

e lo schermo ti mostrerà:

```
1 2 3 4 5 6 7 8 9 MOLTE VOCI
```

Abbiamo, ora, imparato ad eseguire delle semplici operazioni aritmetiche ed a visualizzarne i risultati. Usiamo queste nuove esperienze per risolvere alcuni semplici problemi.

Introduci nel tuo programma solamente numeri ragionevoli.

Esempi applicativi

Calcoliamo il consumo di un'auto in galloni per miglio.
La formula matematica è:

CONSUMO = DISTANZA (in miglia)÷BENZINA (in galloni)

Assumiamo che la distanza sia 510 miglia e la benzina usata sia stata 20.2 galloni. Ecco l'istruzione scritta in BASIC.

```
PRINT "IL CONSUMO È";510/20,2;'MPG" ⮐
```

Ora vogliamo convertire il risultato in litri al chilometro, per coloro che usano il sistema metrico. Un gallone equivale a 3.8 litri e un miglio a 1.6 chilometri.
Il consumo in litri al chilometro è:

```
PRINT "IL CONSUMO DI BENZINA È"; (20.2*3.8)/(510*1.6);"L PER KM" ⮐
```

Ecco un altro esempio. Data una temperatura in gradi Fahrenheit, l'equivalente in gradi Celsius è dato dalla formula:

VALORE CELSIUS = (VALORE FAHRENHEIT—32)*5/9

Quindi per ottenere l'equivalente in gradi CELSIUS di 79° scrivi:

```
PRINT 79 GRADI F=";(79—32)*5/9;GRADI C ⮐
```

Il risultato è:

```
79 GRADI F = 26.1111111 GRADI C
```

Per completezza, osserva che 5/9 non è stato racchiuso tra parentesi perchè non ha importanza se venga eseguito prima il * o il /.

Riassunto

In questo capitolo, abbiamo imparato ad eseguire le operazioni aritmetiche ed a visualizzare testo e risultati sulla stessa linea. Abbiamo usato questa nuova conoscenza per automatizzare il calcolo di semplici formule scrivendole in una singola espressione BASIC.
Finora abbiamo compreso tutti i valori dentro un'unica istruzione BASIC, ciò che vogliamo fare è, prima, scrivere un programma, poi fornirgli ripetutamente dalla tastiera valori diversi senza doverlo riscrivere di nuovo. Riusciremo a fare ciò con l'aiuto delle *variabili* e questo sarà l'argomento del prossimo capitolo.

Esercizi

3.1: Scrivi un'istruzione BASIC che calcoli:

$$\frac{5+6}{1+2+3}$$

3.2: Scrivi un'istruzione BASIC che calcoli:

$$1+{}^{1}/_{2} \qquad \frac{1}{1+{}^{1}/_{2}}$$

3.3: Scrivi un'istruzione BASIC che calcoli l'equivalente Fahrenheit di 20°.

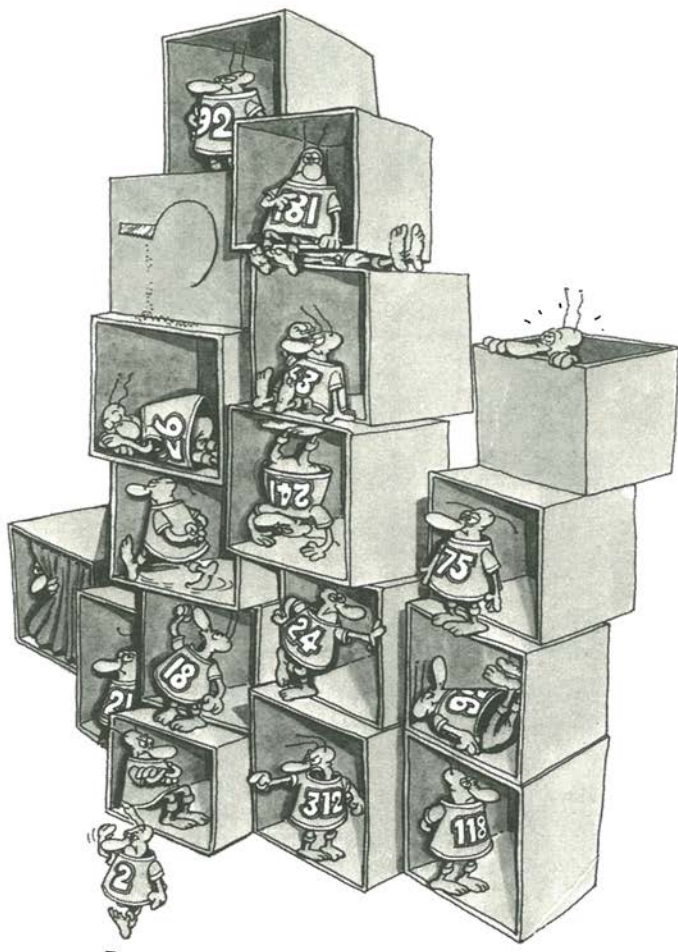
3.4: Data una velocità di 100 Km/h, calcola l'equivalente velocità in miglia all'ora. (1 miglio = 1.6 Km).

3.5: Calcola il numero di secondi in un giorno, in una settimana, in un mese e in un anno.

3.6: Assumendo una velocità media di 55 Km/h, calcola il tempo necessario per fare 350 Km.

3.7: Assumendo che l'anno sia composto di 365 giorni, calcola quanti giorni hai vissuto finora.

3.8: Calcola il salario annuale di una persona, data la:

a. paga settimanale (ci sono 52 settimane in un anno)
b. paga quindicinale (moltiplica per 26)
c. paga mensile
d. paga oraria (moltiplica per 2080)

Capitolo **4**

Memorizzare i valori ed usare le variabili

In questo capitolo, impareremo a scrivere programmi che possano essere usati ripetutamente senza variazioni e che visualizzino risultati diversi in relazione ai dati immessi dalla tastiera. Finora, per ottenere il risultato di una operazione aritmetica, come 2+2, abbiamo dovuto includere nell'istruzione di programma i numeri sui quali operare.
Impareremo ora a scrivere programmi che possano essere eseguiti ogni volta con dati differenti. Il programma specificherà le operazioni da eseguire, ma i dati verranno immessi attraverso la tastiera dall'utente, durante l'esecuzione del programma. Questo rende i programmi riutilizzabili.

Inoltre, introdurremo il concetto di variabile ed impareremo ad usare due nuove istruzioni: INPUT e LET.

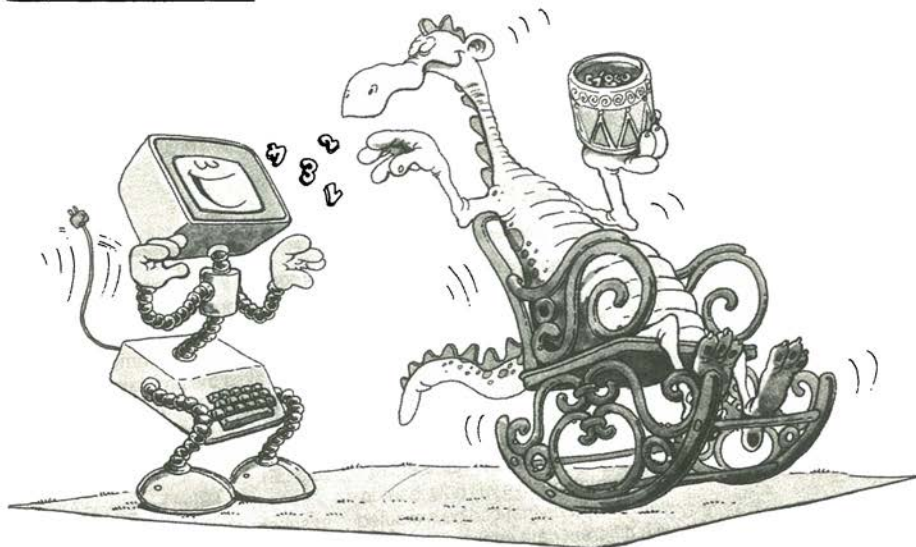
Impariamo ora a dare informazioni al programma durante l'esecuzione.

L'istruzione INPUT

Ricopia il seguente programma. (**Nota bene**: d'ora in avanti non mostreremo più il simbolo per indicare RETURN alla fine di ogni linea):

```
10 INPUT A
20 PRINT A; 2 * A; 3 * A
30 END
```

Ora esegui questo programma col solito comando RUN.
Dovresti vedere sul tuo schermo qualcosa come:

Durante un INPUT,
il tuo computer accetta
sia numeri che lettere.

Generalmente, appare sul tuo schermo un "?" seguito dal cursore lampeggiante per ricordarti di introdurre il dato richiesto.
Ora, scrivi un numero, diciamo 3 e concludi l'introduzione premendo, come al solito, il tasto RETURN. Il tuo schermo dovrebbe ora mostrare:

```
3 6 9
>
```

Il tuo programma è stato eseguito, vediamo cosa è accaduto. La prima linea era:

```
10 INPUT A
```

Questa istruzione ti ha chiesto di introdurre un numero attraverso la tastiera. Il programma ha visualizzato un "?", e si è fermato attendendo il tuo "input" (introduzione dati). Il valore 3 che tu hai fornito è stato letto e memorizzato in A. "A" è chiamata *variabile* ed è un nome usato per memorizzare un valore. Formalmente, una *variabile* è un nome dato ad una locazione di memoria. Alcuni esempi di nomi di variabili sono: A, B, C, F, Z1, G2. La maggior parte dei BASIC permettono anche di usare dei nomi con parecchie lettere, per esempio: NUMERO1, SOMMA, TASSA, RISULTATO.

La seconda istruzione era:

```
20 PRINT A; 2*A; 3*A
```

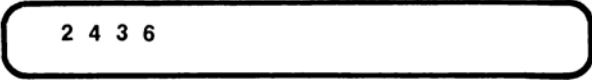
Questa istruzione dà come risultato la visualizzazione di 3, 2*3, 3*3 o:

```
3 6 9
```

Usando l'istruzione INPUT è possibile introdurre anche più di un valore, contemporaneamente. Ecco un esempio:

```
10 INPUT A,B
20 PRINT A;  A * 2; B; B * 2
30 END
```

Ora manda in esecuzione questo programma con RUN. Dovresti vedere il solito "?" apparire sul tuo schermo. Scrivi, allora, due numeri, diciamo 2 e 3, separati da una virgola e poi premi RETURN. Sul tuo schermo dovresti vedere questa volta:

```
2 4 3 6
```

Esaminiamo cosa è accaduto. La prima linea del programma era:

10 INPUT A,B

Il risultato di questa istruzione è stata una richiesta di introduzione di due numeri che sono stati memorizzati nelle variabili A e B. Ricordati che le variabili sono nomi di locazioni di memoria. Il contenuto di A e B era inizialmente vuoto, ma, poi, è divenuto rispettivamente, 2 e 3. La seconda istruzione del programma era:

20 PRINT A; A*2; B; B*2

Questa istruzione stampa i valori 2,2*2,3,3*2 o:

Prova a far funzionare di nuovo questo programma. Prova, ora, scrivendo:

5,8

ed il risultato dovrebbe essere:

```
5 10 8 16
```

Noi possiamo usare questo programma ripetutamente ed ottenere nuovi risultati, inserendo nuovi valori attraverso la tastiera. Abbiamo reso il nostro programma riutilizzabile, usando le variabili (A e B) invece dei valori espliciti.
Per poter rendere questo programma veramente riutilizzabile, miglioriamo lo stile e la leggibilità.
Supponiamo di salvarlo (memorizzarlo in una memoria non-volatile come una cassetta) e rieseguirlo a distanza di parecchi giorni. Potremmo dimenticare come funziona o potremmo non ricordare quanti valori dobbiamo introdurre.
Possiamo riscrivere, allora, il programma in modo che visualizzi queste informazioni sullo schermo. Eccone una versione migliorata:

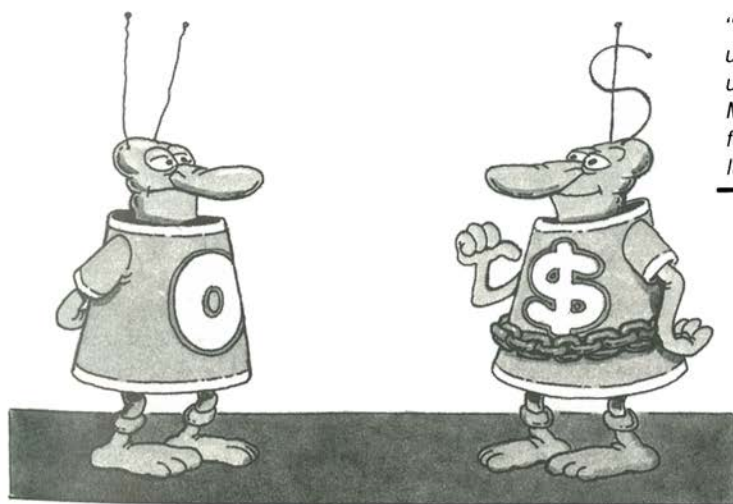
```
10 PRINT "*QUESTO PROGRAMMA MOLTIPLICA
   QUALSIASI COPPIA DI NUMERI PER 2*"
20 PRINT "SCRIVI DUE NUMERI"
30 INPUT A,B
40 PRINT "PRIMO NUMERO : ";A, "RADDOPPIO : "; 2 * A
50 PRINT "SECONDO NUMERO : ";B, "RADDOPPIO : "; 2 * B
60 END
```

ed ecco il risultato visualizzato sullo schermo: (**Nota bene**: in questo libro, i dati introdotti dall'utente verranno scritti in **neretto**).

```
*QUESTO PROGRAMMA MOLTIPLICA QUALSIASI
COPPIA DI NUMERI PER 2*
SCRIVI DUE NUMERI
?5,7
PRIMO NUMERO : 5             RADDOPPIO : 10
SECONDO NUMERO : 7           RADDOPPIO : 14
```

Abbiamo appena imparato come fornire dati numerici al programma usando l'istruzione INPUT. Abbiamo anche introdotto il concetto di variabile. Impariamo ora ad usare effettivamente queste tecniche ed a sviluppare programmi più complessi.

I due tipi di variabile

Ci sono due tipi di variabili: *numeriche* e *stringhe*. Le variabili numeriche rappresentano numeri e le variabili stringhe rappresentano testi. Questi due tipi di variabili hanno una differenza: una variabile stringa ha una "$" alla fine del nome. Esse si usano in modo diverso; ad esempio, si puo fare la somma di due numeri, ma non di due testi. Impareremo prima qualcosa sulle variabili numeriche e, in seguito, sulle variabili stringa.

"Ehi, Numero, io sono una stringa. Io conservo una catena di caratteri. Mi puoi riconoscere facilmente ... guarda la mia antenna!"

Variabili numeriche

Impariamo le regole che governano i nomi delle variabili numeriche e poi impareremo ad usare effettivamente queste variabili. Abbiamo già usato due variabili numeriche, chiamate A e B, all'inizio di questo capitolo ed abbiamo assegnato loro un valore attraverso la tastiera tramite l'istruzione INPUT. Impariamo ora come dare un nome ad una variabile.
Per assegnare un nome ad una variabile, tutti i BASIC (inclusi l'originale Dartmouth BASIC) permettono l'uso di una lettera, seguita opzionalmente da un singolo numero. Ecco degli esempi di nomi validi di variabili:

A	*(una lettera)*
B	*(una lettera)*
Z	*(una lettea)*
A1	*(una lettera ed un numero)*
A2	*(una lettera ed un numero)*
B2	*(una lettera ed un numero)*

Mentre con questa definizione, i seguenti nomi non sono permessi:

I2	*(non inizia con una lettera)*
A2B	*(troppi caratteri)*
BA	*(è permessa solo una lettera)*
1B	*(deve iniziare con una lettera)*
AB1	*(troppo lungo, è permessa solo una lettera)*

Il vantaggio dei nomi corti è la riduzione della dimensione e della complessità dell'interprete. Lo svantaggio è che i nomi corti sono difficili da ricordare.
Per esempio, il nome lungo RISULTATO è molto più descrittivo e facile da ricordare del nome corto R. Per migliorare la leggibilità, la maggior parte dei nuovi BASIC permettono i nomi lunghi, cioè le sequenze di caratteri.
Generalmente puoi usare qualsiasi numero di lettere consecutive seguite da numeri opzionali fino ad una lunghezza massima. Per esempio, i seguenti nomi di variabili sono corretti:

VINCITORE	STUDENTE 1
PERDENTE	STUDENTE 2
RISULTATO	STUDENTE 14
SOMMA	CASO 24

I seguenti, invece, non sono permessi:

R2D2	*(le lettere non sono consecutive)*
3 VOLTE	*(inizia con una cifra)*
A-UNO	*(c'è un simbolo non permesso)*

Ogni variabile deve avere un nome.

Chiaramente un programma che usa i nomi delle variabili lunghi ed espliciti è molto più leggibile. In questo capitolo, useremo sia i nomi lunghi che quelli corti, affinché tu possa usare entrambe le convenzioni. Ricordati che i nomi lunghi sono semplicemente una comodità, ma non influiscono in alcun modo sul programma.

C'è ancora una restrizione sui nomi; non puoi usare le parole riservate, cioè i nomi che hanno un significato per l'interprete BASIC. Per esempio, non puoi usare come nomi di variabili, parole come LIST, END o RUN.
(**Nota bene**: un elenco delle più comuni parole riservate del BASIC compare alla fine del libro, così come compare alla fine del manuale di riferimento del tuo interprete BASIC).
Alcuni BASIC, inoltre, proibiscono di usare le parole riservate come *parte* di un nome. Per esempio, OR è una parola riservata standard; perciò con alcuni BASIC non potresti usare la parola PORTA come nome di variabile.
Ora che sappiamo come creare correttamente i nomi delle variabili, impariamo come dare un nome ad un pezzo di testo detto *stringa*.

Variabili stringa

Inizialmente, introduciamo le stringhe; eccone alcuni esempi:

"RISULTATO"
"QUESTO È UN ESEMPIO"
"IL MIO NOME È GIANNI"
"25 PER 4 ="

Osserva che una stringa è normalmente contenuta tra le virgolette; altrimenti, potrebbe essere confusa con un nome di variabile. Una stringa può contenere qualunque sequenza di caratteri ad eccezione delle virgolette. (Alcune versioni del BASIC permettono, comunque, di aggirare questa limitazione). La lunghezza di una stringa è sempre limitata. Un limite tipico può essere 128 caratteri.
Introduciamo ora le variabili stringa. Quando il valore memorizzato in una variabile è un testo (cioè una stringa) piuttosto che un numero, la variabile si chiama *variabile stringa*. Un nome di una variabile stringa è simile ad un nome di una variabile numerica ad eccezione che possiede un "$" alla fine. Ecco alcuni nomi di variabile stringa corti:

A$
R$
A1$
B5$

e se il tuo BASIC permette l'uso di nomi lunghi, eccone altri, altrettanto validi:

NOME$
PRIMA$
CITTA'$
UNITA' 25$

Ricorda che un nome come BRANDA$ può non essere permesso con alcuni BASIC perchè contiene la parola riservata AND. Comunque non te ne preoccupare, perchè l'interprete ti avvertirà subito quando la scriverai.
Illustriamo l'uso delle variabili stringa, con un programma che saluta l'utente con il suo nome.

```
10 PRINT "IO SONO HAL, IL COMPUTER"
20 PRINT "QUAL'E' IL TUO NOME ";
30 INPUT NOME$
40 PRINT "E QUAL'E' IL TUO COGNOME";
50 INPUT COGNOME$
60 PRINT "SALVE, "; NOME$;COGNOME$;"!"
70 PRINT "ORA CONOSCO IL TUO NOME!"
80 PRINT "MI PIACE  "; NOME$;COMENOME."
90 END
```

"Io sono una variabile stringa. Contengo testi e il mio nome termina con una $".

Se il tuo BASIC richiede i nomi corti, usa N$ al posto di NOME$ e C al posto di COGNOME$.
Ecco una semplice esecuzione. Osserva che i caratteri che tu introduci sono in neretto.

```
RUN
IO SONO HAL, IL COMPUTER
QUAL'E' IL TUO NOME ? PAOLO
E QUAL'E' IL TUO COGNOME ? ROSSI
SALVE, PAOLO ROSSI!
ORA CONOSCO IL TUO NOME!
MI PIACE PAOLO COME NOME.
>
```

Ora sei in grado di comunicare con il tuo computer.
Puntualizziamo su alcune caratteristiche basilari di questo programma. Cominciamo con la linea 20:

20 PRINT "QUAL'È IL TUO NOME";

Osserva che questa linea termina con un punto e virgola.
Il punto e virgola significa "visualizza il prossimo carattere subito dopo questo testo".

Il risultato dell'istruzione 20 e della tua risposta all'istruzione 30 è:

QUAL'È IL TUO NOME? PAOLO

e se tu avessi scritto:

20 PRINT "QUAL'È IL TUO NOME"

senza il punto e virgola finale, il risultato sarebbe stato:

QUAL'È IL TUO NOME

? PAOLO

Naturalmente, puoi scegliere entrambi i metodi; la forma è materia di scelta, ma ricordati: se desideri che il carattere che introduci appaia sulla stessa linea del precedente messaggio, usa un punto e virgola alla fine dell'istruzione PRINT; altrimenti, la prossima posizione sarà all'inizio della linea seguente.
Ora che ne sappiamo qualcosa di più sulle variabili numeriche e stringa, usiamole per continuare il nostro dialogo con Hal, il computer. Aggiungiamo quanto segue al nostro programma:

```
90 PRINT "QUAL'E' L'ANNO CORRENTE (2 CIFRE)";
100 INPUT ANNOCORRENTE
110 PRINT "IN QUALE ANNO SEI NATO (2 CIFRE)";
120 INPUT ANNONASCITA
130 PRINT "CARO ";NOME$;", QUEST'ANNO HAI O AVRAI"
    ;ANNOCORRENTE-ANNONASCITA;"ANNI"
140 END
```

(**Nota bene**: Alcuni interpreti potrebbero non accettare la variabile ANNOCORRENTE, perchè contiene la parola riservata OR. In questo caso usa un nuovo nome, come ANNOATTUALE).
Ecco un semplice dialogo. Nuovamente, i caratteri che introduci sono scritti in neretto.

```
QUAL'E' L'ANNO CORRENTE (2 CIFRE) ? 82
IN QUALE ANNO SEI NATO (2 CIFRE) ? 50
CARO PAOLO, QUEST'ANNO HAI O AVRAI 32 ANNI
```

Esaminiamo dettagliatamente il programma. L'istruzione

90 PRINT "QUAL'È L'ANNO CORRENTE (2 CIFRE)";

stampa un evidente messaggio e, di nuovo, è stato usato il punto e virgola finale per scrivere le due cifre sulla stessa linea. Nella istruzione successiva.

100 INPUT ANNOCORRENTE

ANNOCORRENTE è una variabile numerica. Il valore 82 che hai scritto è stato memorizzato in essa e, d'ora in avanti, quando viene usato il nome ANNOCORRENTE, esso viene sostituito automaticamente dall'interprete con il valore 82. Ciò accade nell'istruzione 130.

L'istruzione

110 PRINT "IN QUALE ANNO SEI NATO (2 CIFRE)";

è simile all'istruzione 90. Osserva il punto e virgola finale. L'istruzione

120 INPUT ANNONASCITA

è simile all'istruzione 100. ANNONASCITA è una variabile numerica che conterrà il valore 50.
Nell'istruzione seguente usiamo questa variabile ed il valore 50 viene sostituito automaticamente dall'interprete.
Esaminiamo questa linea:

130 PRINT "CARO"; NOME$; ",QUEST'ANNO HAI O AVRAI";
ANNOCORRENTE-ANNONASCITA; "ANNI"

Quando questa istruzione verrà eseguita, verrà visualizzato quanto segue:

CARO PAOLO QUEST'ANNO HAI O AVRAI 32 ANNI

Dividiamo il testo per esaminarlo.

CARO

(Questa si chiama stringa letterale. Essa è una stringa che fa parte di una istruzione).

PAOLO

(Questa è il valore della stringa letto dalla tastiera e memorizzato nella variabile chiamata NOME$. Essa rimarrà così finché non verrà usato il comando NEW o verrà memorizzato un nuovo valore in NOME$).

QUEST'ANNO HAI O AVRAI

(Questa è un'altra stringa letterale).

32

(Questo è il risultato di ANNOCORRENTE-ANNONASCITA, cioè 82-50 = 32).

ANNI

(Un'altra stringa letterale).

Durante la scrittura e l'esame di questo programma, potresti già aver avuto qualche delusione. Per esempio, potresti aver desiderato introdurre la data attuale compresa di giorno e di mese, così da calcolare l'età alla data odierna. Questo però richiederebbe di confrontare la data e il mese odierno con quello della tua nascita e per far questo, dobbiamo ancora imparare una nuova istruzione BASIC:

IF (condizione) THEN (esegui)

che tradotta letteralmente significa "SE è vera la condizione ALLORA esegui questa istruzione".
Ma questo argomento verrà trattato nel Capitolo 7.
Un'altra cosa che potresti desiderare è di scrivere un programma che possa essere usato ripetutamente (senza dover usare ogni volta il RUN). Impareremo a far questo quando, nel capitolo 6, tratteremo l'istruzione GOTO (in inglese significa "vai a").
Ora che abbiamo preso familiarità sia con le variabili numeriche che con le stringhe, impareremo ad usarle in un programma più lungo, prima assegnando un valore alla variabile, poi usando la tecnica dei contatori.

Assegnare un valore ad una variabile
(L'istruzione LET)

Finora abbiamo assegnato un valore ad una variabile, usando esclusivamente l'istruzione INPUT. Per esempio, quando è stata eseguita l'istruzione:

```
20 INPUT A
```

tu hai introdotto un valore, come 5.2 (seguito da $) e il valore di A è divenuto 5.2. Esiste comunque, un'altra istruzione per assegnare un valore ad A, che è chiamata istruzione di *assegnazione*. Eccone un esempio:

```
10 A = 52
```

(**Nota bene**: se possiedi un Integer BASIC, allora usa 5 al posto di 5.2). Questa istruzione assegna il valore 5.2 alla variabile A come parte di programma, senza, quindi, dover introdurre il dato dalla tastiera. Puoi anche scrivere:

```
10 B = 1
20 C = 2
30 A = B + C
```

Come puoi vedere, con l'esecuzione dell'istruzione 30, ad A viene assegnato il valore 2+1 = 3.

"Ehi, Z, desidero che tu tenga questo valore!"

Nei primi BASIC, l'istruzione di assegnazione doveva iniziare con la parola riservata LET (poni). In tali BASIC, l'esempio precedente dovrà essere scritto:

```
10 LET B = 1
20 LET C = 2
30 LET A = B + C
```

Lo scopo dell'istruzione LET era di semplificare l'interprete dicendogli chiaramente che l'istruzione era una assegnazione. L'istruzione LET generalmente ora non si usa più; senza di essa il programmatore deve scrivere un po' meno, quando formula un'istruzione d'assegnazione.
Mostriamo ora il valore delle istruzioni di assegnazione, esaminando due esempi. In entrambi calcoleremo la somma e la media di due numeri. Ecco il primo programma senza le istruzioni di assegnazione:

```
10 PRINT "DAMMI DUE NUMERI"
20 PRINT "CALCOLERO' LA LORO SOMMA E LA MEDIA"
30 PRINT "IL PRIMO NUMERO, PER PIACERE:";
40 INPUT A
50 PRINT "IL SECONDO NUMERO, PER PIACERE:";
60 INPUT B
70 PRINT "LA SOMMA DI"; A; "PIU'"; B; "E': "; A + B
80 PRINT "LA LORO MEDIA E':"; (A + B)/2
90 END
```

ed ecco lo schermo dopo il RUN:

```
>RUN
DAMMI DUE NUMERI
CALCOLERO' LA LORO SOMMA E LA MEDIA
IL PRIMO NUMERO, PER PIACERE:? 24
IL SECONDO NUMERO, PER PIACERE:? 41
LA SOMMA DI 24 PIU' 41 E': 65
LA LORO MEDIA E': 32.5
>
```

Nota che l'espressione A+B viene ripetuta due volte nell'istruzione PRINT, ma questo è l'inconveniente minore anche se, in un programma più lungo, ciò aumenterebbe la probabilità di commettere errori di scrittura. Inoltre se volessimo utilizzare lo stesso programma con una formula diversa, ne dovremmo riscrivere una grossa parte.
Qui, di seguito, c'è un programma equivalente che usa una *variabile intermedia*, chiamata SOMMA, per memorizzare il risultato. Questo nuovo programma è molto più leggibile e meno propenso ad errori.

```
10  PRINT "DAMMI I NUMERI"
20  PRINT "CALCOLERO' LA LORO SOMMA E LA MEDIA"
30  PRINT "IL PRIMO NUMERO, PER PIACERE:";
40  INPUT A
50  PRINT "IL SECONDO NUMERO, PER PIACERE:";
60  INPUT B
70  SOMMA= A + B
80  MEDIA = SOMMA/2
90  PRINT "LA SOMMA DEI NUMERI E':"; SOMMA
100 PRINT "LA LORO MEDIA E':"; MEDIA
110 END
```

Sono state usate due nuove variabili:

```
70 SOMMA = A+B
80 MEDIA = SOMMA/2
```

Usando le variabili addizionali, si ottengono due vantaggi: i programmi sono più chiari e sono più facilmente modificabili. Per esempio, ammettiamo di voler modificare il programma per fare la media di tre numeri. Per far questo, dovremmo scrivere:

```
62 PRINT "IL TERZO NUMERO, PER PIACERE:";
64 INPUT C
70 SOMMA = A+B+C
80 MEDIA = SOMMA/3
```

e potremmo lasciare il resto inalterato. Abbiamo semplicemente letto il terzo numero e modificato le formule solamente in un punto. Ecco il programma completo:

```
10  PRINT "DAMMI I NUMERI"
20  PRINT "CALCOLERO' LA LORO SOMMA E LA MEDIA"
30  PRINT "IL PRIMO NUMERO, PER PIACERE:";
40  INPUT A
50  PRINT "IL SECONDO NUMERO, PER PIACERE:";
60  INPUT B
62  PRINT "IL TERZO NUMERO, PER PIACERE:";
64  INPUT C
70  SOMMA= A + B + C
80  MEDIA = SOMMA/3
90  PRINT "LA SOMMA DEI NUMERI E':"; SOMMA
100 PRINT "LA LORO MEDIA E':"; MEDIA
110 END
```

e lo schermo dopo il RUN:

```
DAMMI I NUMERI
CALCOLERO' LA LORO SOMMA E LA MEDIA
IL PRIMO NUMERO, PER PIACERE:? 5
IL SECONDO NUMERO, PER PIACERE:? 3
IL TERZO NUMERO, PER PIACERE:? 10
LA SOMMA DEI NUMERI E': 18
LA LORO MEDIA E': 6
```

Abbiamo imparato due metodi per assegnare un valore ad una variabile:

- Possiamo usare un'istruzione INPUT quando il valore va introdotto durante l'esecuzione del programma.
- Possiamo usare l'istruzione di assegnazione quando il valore (o l'espressione) va memorizzato insieme con il programma.

Il primo metodo (istruzione INPUT) dovrebbe essere usato quando ti aspetti che il valore da assegnare alla variabile esplicitamente (cioè non un valore *calcolato*), sia diverso ogni volta che il programma venga eseguito.

Il secondo metodo (usando l'istruzione d'assegnazione) dovrebbe essere usato quando ti servi di una formula per calcolare il valore della variabile o quando ti aspetti che il valore esplicito (cioè non *calcolato*) rimanga lo stesso ogni volta che il programma venga eseguito.
Impariamo ora l'insieme di regole che governano la scrittura delle istruzioni d'assegnazione.

La sintassi di una istruzione d'assegnazione

Le regole (o sintassi) per scrivere una istruzione d'assegnazione sono semplici. La forma generale è la seguente:

<variabile> = <espressione>

cioè deve esserci sempre una variabile a sinistra e una espressione sulla destra. Più precisamente, un'espressione è:

- un numero o una variabile
- un numero o una variabile, seguita da un operatore (come +, —, *, /) ed un'altra espressione.

Ecco alcune semplici espressioni:

3	(*un numero*)
A	(*una variabile*)
2+2	(*numero, operatore, numero*)
A+2	(*variabile, operatore, numero*)
A+B*3	(*variabile, operatore, espressione*)

Le espressioni possono essere racchiuse tra le parentesi, per esempio:

3+(A+2)/2

B+((C*2)+(D/2))/4

Puoi pensare ad una espressione come ad un valore o come a qualcosa che possa essere calcolato e ne risultasse un valore (in altre parole, come una formula per calcolare un valore).

Il segno uguale (=) usato nell'istruzione d'assegnazione non deve essere interpretato come lo stesso simbolo in matematica. In una espressione esso significa "riceve il valore di". Per esempio, puoi scrivere:

```
10  A = 1
20  A = A + 1
```

(In matematica, l'espressione A=A+1 sarebbe assurda e senza significato). In BASIC, dopo che l'istruzione 20 è stata eseguita, il valore di A è 1 (precedente valore di A) + 1 = 2. Per evitare ogni confusione, molti linguaggi programmativi moderni usano la ← o il simbolo :=, invece del segno =. Ricordati perciò che in una istruzione d'assegnazione in BASIC, il segno = significa che la variabile di sinistra riceve il valore dell'espressione sulla destra.
Ecco alcuni esempi validi di istruzioni d'assegnazione:

A = —3 + 2 (*–3 è un intero negativo*)

B = A + 1

C = (2*3) + (A/B)

MEDIA = SOMMA/NUMERO

QUADRATO = A ** 2

X = B ** 2 — (4*A*C)

Esaminiamo ora l'ultima assegnazione e verifichiamo che risponda alla nostra definizione:

B ** 2

<variabile> <operatore> <numero>

seguito da

—(4*A*C)

cioè

<operatore> <espressione con parentesi equivalente ad un valore>

Dentro le parentesi:

4*A*C

è

<numero> <operatore> <variabile> <operatore> <variabile>

Si, è proprio una espressione corretta.

Le seguenti espressioni, invece non sono corrette:

B + C = SOMMA	(*alla sinistra del segno =, può esserci solo una variabile (non una espressione)*)
2 = A	(*a sinistra deve esserci una variabile e non un numero*)
SOMMA = B + C (D/3)	(*manca l'operatore dopo C*)
(MEDIA) = (B + C)/2	(*a sinistra non devono esserci parentesi*)
A =	(*manca il valore sulla destra*)

Per finire, osserva che tutte le variabili usate a destra di una istruzione di assegnazione devono avere già ricevuto un valore. Se scrivi:

```
10 B = 2
20 SOMMA = B + C
30 INPUT C
```

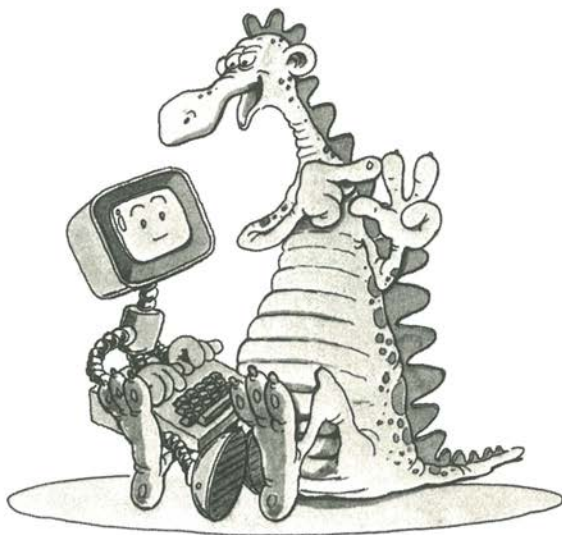
il programma commetterà un errore alla linea 20, perchè C non aveva un valore assegnato. Probabilmente intendevi scrivere:

```
10 B = 2
20 INPUT C
30 SOMMA = B + C
```

Ora che abbiamo imparato la sintassi dell'assegnazione, usiamo questa nuova conoscenza per introdurre una importante tecnica che usa l'assegnazione: la *tecnica dei contatori*.

La tecnica del "contatore"

Ricordati che una variabile è semplicemente un nome dato ad una locazione di memoria. Un valore può essere memorizzato in una locazione usando l'istruzione INPUT o un'istruzione d'assegnazione (=). In questo esempio, noi vogliamo cambiare ripetutamente il valore di una variabile per contare degli eventi. Questa tecnica che noi useremo, è chiamata tecnica del *contatore*.

"Divertiamoci a contare!"

"Io sono un contatore".

Ora dimostriamo come successive assegnazioni possano cambiare il contenuto della variabile N. Scrivi, quanto segue, nel modo immediato. (**Nota bene**: Questo modo è chiamato anche modo di *calcolo*):

N = 1

Il valore sarà automaticamente memorizzato in N.

Verifichiamolo, scrivendo:

PRINT N

Apparirà il valore 1. Ora scrivi:

N = 2

N ora contiene il valore 2. Scrivi:

PRINT N

La risposta sarà

2

Il valore 2 ha sostituito il valore 1 in N. scrivi:

N = 3

Ora scrivi:

PRINT N

e verifica che il valore 3 sia memorizzato in N. La cosa quindi funziona e noi useremo questa tecnica per contare degli eventi. In altre parole, una variabile può essere usata come contatore di eventi. Di seguito, c'è un'anteprima di un programma avanzato che conta quante volte introduci un numero alla tastiera e si ferma quando premi lo zero.

```
10 SOMMA = 0
20 SOMMA = SOMMA + 1
30 PRINT "INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE";
40 INPUT NUMERO
50 PRINT "HAI INTRODOTTO"; SOMMA; "NUMERI"
60 IF NUMERO <> 0 THEN 20
70 END
```

Più tardi, nel Capitolo 6, esamineremo in dettaglio istruzioni come la numero 60. Questa istruzione significa "se (IF) NUMERO non è uguale a zero, allora (THEN) esegui l'istruzione 20. Ecco una esecuzione tipica:

```
INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE? 5
HAI INTRODOTTO 1 NUMERI
INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE? 1
HAI INTRODOTTO 2 NUMERI
INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE? 2
HAI INTRODOTTO 3 NUMERI
INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE? 3
HAI INTRODOTTO 4 NUMERI
INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE? 4
HAI INTRODOTTO 5 NUMERI
INTRODUCI UN NUMERO. SCRIVI 0 PER SMETTERE? 0
HAI INTRODOTTO 6 NUMERI
```

In questo programma, il valore di SOMMA è *inizializzato* a 0 nella prima istruzione ed è incrementato di uno, ogni volta viene introdotto un numero.
Questo è un contatore. Vedremo molti altri esempi di questa tecnica quando scriveremo altri programmi. In seguito, impareremo a mettere a punto questo programma, se non vogliamo che 0 conti come numero quando fermiamo l'esecuzione.

Riassunto

In questo capitolo, abbiamo imparato a scrivere programmi che possano essere usati ripetutamente. Questi programmi forniranno risultati diversi a seconda dei valori introdotti alla tastiera. Abbiamo raggiunto questo scopo usando le variabili ed assegnando loro i valori in vari modi. Una variabile dovrebbe essere pensata come un nome dato ad una locazione di memoria nella quale possa essere memorizzato un valore o un testo.
Abbiamo imparato anche a cambiare il contenuto di una variabile tramite una istruzione INPUT o un'assegnazione.
Siamo ora in grado di scrivere semplici programmi che automatizzino il dialogo o un semplice calcolo.
Inoltre, i nostri programmi, ora, superano considerevolmente le dieci linee: sono diventati lunghi.
Scrivili sempre con chiarezza. Nel prossimo capitolo, prima di procedere oltre e di imparare nuovi strumenti e tecniche, impareremo a scrivere un programma in modo chiaro.

Esercizi

4.1: Leggi dalla tastiera quattro numeri e visualizza il totale, la media e il prodotto dei quattro numeri.

4.2: Presumendo che il tuo BASIC permette l'uso di variabili con il "nome lungo", sono permessi i nomi seguenti?

a. 24B	e. ALFA2D	i. PI
b. B24	f. ESEMPIO	j. 3$
c. A+B	g. INPUT	k. TRE
d. APIUB	h. INPUT1	l. NOME$

4.3: Scrivi un programma che chieda il nome dell'utente e dica: "MI SEMBRA DI CONOSCERE UN (qui il nome)!"

4.4: Scrivi un programma che richieda:

— il nome di un oggetto
— il nome di un mobile
— il nome di un amico

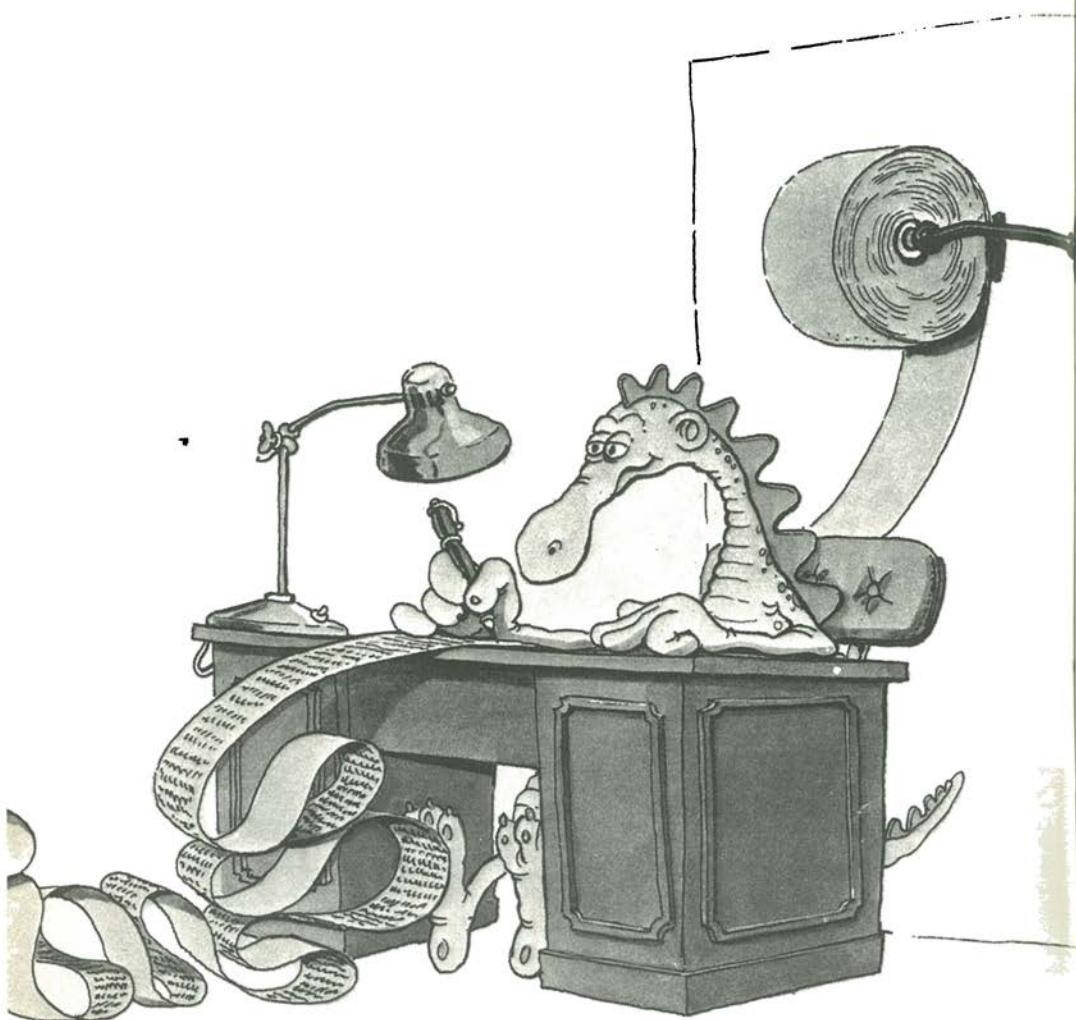
e dica: "IL TUO AMICO (nome) HA UN (oggetto) SOPRA UN (mobile)?"

4.5: Scrivi un programma che richieda il colore dei tuoi occhi e dica: "MI PIACCIO-NO GLI OCCHI (colore)".

4.6: Sono permesse le seguenti assegnazioni?

a. A + 1 = A d. B + C = A

b. A = A + A + A e. 3 = 2 + 1

c. A = B + C f. NUMERO = PRIMO + ULTIMO * 2

Capitolo **5**

Scrivere un programma in modo chiaro

Finora abbiamo scritto dei programmi brevi, usando tre tipi differenti di istruzioni: PRINT, INPUT e l'assegnazione (=) ed abbiamo anche imparato a scrivere delle brevi espressioni aritmetiche. Nel capitolo seguente, impareremo delle nuove tecniche e dei nuovi tipi d'istruzioni e scriveremo programmi più lunghi. Prima di farlo, però, impareremo a rendere i nostri programmi chiari e leggibili.
Rendere un programma leggibile è importante perchè se scrivi un programma oggi e lo userai o lo vorrai modificare tra qualche giorno, perderai probabilmente molto tempo per ricordarti e capire come funziona e cosa fa. Come regola, per prima cosa studia ogni programma nei minimi particolari e poi, quando lo scrivi, rendilo più leggibile che puoi.
Lo scopo di questo capitolo è di aiutarti a migliorare la leggibilità dei tuoi programmi.

Osserveremo sette tecniche:

1. L'uso della istruzione REM. (Per introdurre dei *commenti* (remark) sui programmi).
2. L'uso di più istruzioni sulla stessa linea.
3. L'uso di spazi dentro una istruzione.
4. L'uso di istruzioni "PRINT vuote" e CLE (per migliorare la visualizzazione sullo schermo).
5. L'uso di istruzioni "INPUT abbreviate" (per ridurre il numero delle linee in un programma).
6. La scelta di nomi significativi per le variabili.
7. Una opportuna numerazione delle linee.

Ora descriveremo queste tecniche una dopo l'altra.

L'istruzione REM

Ecco un esempio che usa l'istruzione REM:

```
10 REM * * * PROGRAMMA DI ADDIZIONE * * *
20 PRINT "DAMMI DUE NUMERI:";
30 INPUT PRIMO,ULTIMO
40 PRINT "LA LORO SOMMA E':"; PRIMO + ULTIMO
50 END
```

L'istruzione REM è usata per rendere più leggibile il programma, introducendo dei commenti nel testo. Questa istruzione viene ignorata dall'interprete durante l'esecuzione del programma e non ha nessuna conseguenza sul programma stesso. Ecco altri esempi di commenti che potresti usare:

```
25 REM ORA LEGGI DUE NUMERI
45 REM POTREMMO ANCHE MOLTIPLICARLI NELLA ISTRUZIONE PRECEDENTE
```

e ecco il programma che ne risulta:

```
10 REM * * * PROGRAMMA DI ADDIZIONE * * *
20 PRINT "DAMMI DUE NUMERI:";
25 REM ORA LEGGI I DUE NUMERI
30 INPUT PRIMO,ULTIMO
40 PRINT "LA LORO SOMMA E':"; PRIMO + ULTIMO
45 REM POTREMMO ANCHE MOLTIPLICARLI
   NELLA ISTRUZIONE PRECEDENTE
50 END
```

I commenti REM sono invisibili all'interprete.

Per migliorare la leggibilità, usa liberamente asterischi, lineette o altri simboli:

```
10  REM * * * PROGRAMMA DI ADDIZIONE * * *
60  REM ----- SECONDA PARTE -----
100 REM = = = = = GRAN FINALE = = = = =
200 REM $$$$$CAMBIA IN SEGUITO QUESTA PARTE$$$$$
```

"Guarda come pochi asterischi rendono il mio programma più leggibile".

Istruzioni multiple sulla stessa linea

La maggior parte dei BASIC permettono di inserire due o più istruzioni sulla stessa linea, separandole con i due punti. Ecco un esempio:

```
50  PRINT "SCRIVI UN NUMERO"; : INPUT NUMERO
```

che è equivalente a

```
50  PRINT "SCRIVI UN NUMERO";
60  INPUT NUMERO
```

Osserva che deve esserci un solo numero di linea per ogni linea, perciò quando due istruzioni sono scritte sulla stessa linea, c'è solo un numero di linea sulla sinistra. Ecco un altro esempio:

```
100  REM***CALCOLA TUTTO SU UNA SOLA LINEA***
110  SOMMA=A+B:PRODOTTO=A*B:MEDIA=SOMMA/2
```

Sono presenti tre istruzioni sulla linea 110.
La scrittura di istruzioni multiple è utile almeno in due casi: per rendere chiari gli INPUT e per inserire dei commenti (REM) alla destra di una istruzione: Ecco un esempio di come rendere più chiaro un INPUT:

```
70  PRINT "INTRODUCI 2 NUMERI"; : INPUT N1, N2
```

La linea 70 è scritta così come opera sullo schermo. Questo rende più facile la lettura. Ecco un esempio che mostra come un commento è introdotto sulla stessa linea a cui si riferisce:

```
60  RISULTATO=A+2*B—5:REM IL RISULTATO DEVE ESSERE POSITIVO
```

Eccone un altro:

```
50  TEMPERATURA=(FAHR—32)*5/9:REM CONVERSIONE A CELSIUS
```

L'uso degli spazi bianchi

Eccetto che nei nomi, nelle stringhe e nei dati in ingresso, gli spazi bianchi sono generalmente ignorati dal BASIC. Per esempio, puoi scrivere:

```
20  PRINT4+2*3
```

Comunque, una istruzione come questa è poco leggibile. Per facilitare la lettura, usa liberamente gli spazi bianchi. Usali:

▸ *dopo ogni parola riservata, come PRINT o INPUT.*

Ecco degli esempi:

```
50  PRINT 4
60  INPUT NUMERO
```

▸ *prima o dopo ogni operatore*

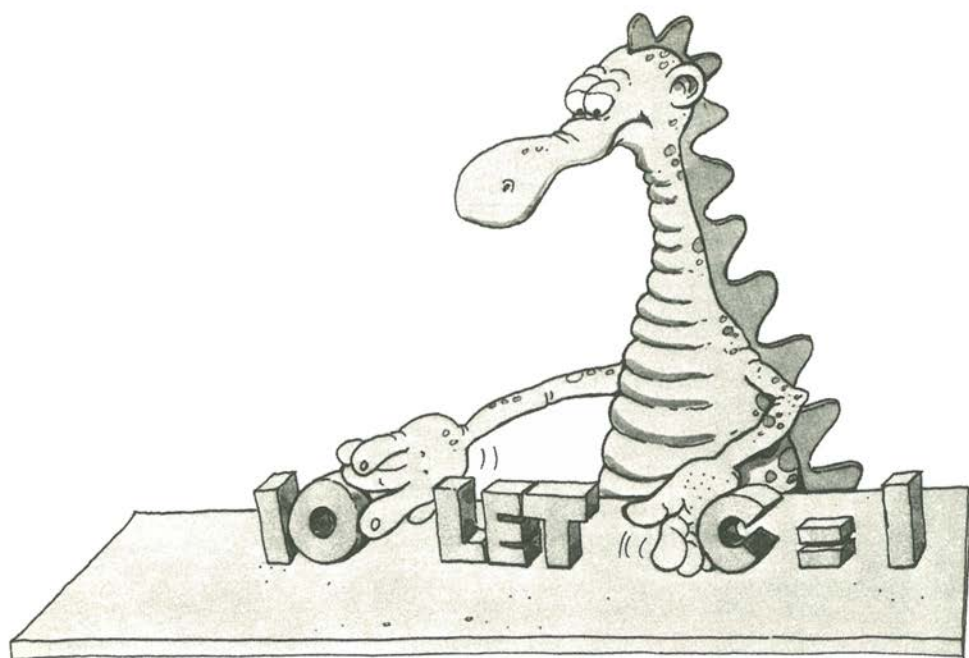
Ecco degli esempi:

```
30  PRINT 4 + 2 + 3
40  RISULTATO = A1 / ((B — C) * D)
```

Puoi anche usare gli spazi prima delle parole riservate per allineare le istruzioni nel tuo programma. Eccone un esempio:

```
10  PRINT "DIECI"
20  PRINT "VENTI"
90  PRINT "NOVANTA"
100 PRINT "CENTO"
200 PRINT "DUECENTO"
```

che senza i due spazi iniziali nelle linee 10,20 e 90, apparirebbe:

```
10 PRINT "DIECI"
20 PRINT "VENTI"
90 PRINT "NOVANTA"
100 PRINT "CENTO"
200 PRINT "DUECENTO"
```

Usa gli spazi senza avarizia.

Per finire, ecco esempio di come usare gli spazi all'interno di una istruzione REM per facilitarne ancora di più la leggibilità:

```
1 REM QUESTO PROGRAMMA TIENE IL MIO INVENTARIO
2 REM COPYRIGHT MIO 1982
3 REM QUESTE SONO LE VARIABILI:
4 REM C   E' IL COLORE (DA 1 A 10)
5 REM U   E' IL NUMERO DELLE UNITA' (FINO A 1000)
6 REM T   E' LA TAGLIA (DA 1 A 50)
7 REM CST E' IL COSTO UNITARIO
8 REM D   E' IL PREZZO AL DETTAGLIO
9 REM Q   E' LA QUANTITA' DI RIORDINO
```

Tieni presente che gli spazi sono usati per migliorare la leggibilità, ma non puoi, comunque, aggiungere spazi dentro una parola riservata, in un nome di variabile o durante la risposta ad un INPUT, a meno che gli spazi non siano parte di una stringa.

Miglioramento della visualizzazione

Due nuove tecniche possono essere usate per migliorare l'aspetto sul tuo schermo: l'istruzione CLEAR (o CLE: pulizia) e l'uso di istruzioni PRINT vuote.
L'istruzione CLEAR pulisce lo schermo ogni volta che viene usata. Puoi iniziare ogni programma con

```
10 NEW : CLE
```

Questa è una doppia istruzione su una singola linea che pulisce la memoria (NEW) e lo schermo (CLEAR).
Ecco un trucco che potrai usare alla fine del tuo programma:

```
CLEAR:LIST
```

Per motivi più o meno alle note precedente, proporrei di eliminare il numero di linee da questa istruzione.

Questa doppia istruzione cancellerà lo schermo e listerà l'intero programma (prima che venga eseguito).
Molti BASIC, ma non tutti, possiedono l'istruzione CLE o un comando simile (CLS, per esempio). Attento che il comando CLE nel BASIC Microsoft azzera le variabili, piuttosto che pulire lo schermo. (**Nota bene**: se il tuo BASIC non possiede un comando CLE, puoi generalmente ottenere lo stesso risultato scrivendo:

```
10 PRINT CHR$(NUM)
```

dove NUM contiene il numero di uno speciale carattere (specificato sul manuale del tuo display) che pulirà lo schermo).
L'istruzione PRINT vuota può essere usata per visualizzare una linea vuota. Se scrivi:

```
50  PRINT
```

verrà visualizzata una linea vuota. Se desideri saltare tre linee sullo schermo, scrivi:

```
50  PRINT : PRINT : PRINT
```

o, equivalentemente:

```
50  PRINT
60  PRINT
70  PRINT
```

INPUT abbreviato

Ogni volta che usi una istruzione INPUT, dovresti avvertire l'utente del programma, spiegandogli che cosa introdurre. Ecco un esempio:

```
50  PRINT "SCRIVI DUE INTERI"
60  INPUT ETA1, ETA2
```

La sequenza PRINT-INPUT è talmente usata che la maggior parte dei BASIC moderni permette l'uso di un INPUT abbreviato. Puoi scrivere:

```
50  INPUT "SCRIVI DUE INTERI"; ETA1, ETA2
```

che è equivalente alle due istruzioni precedenti. Questa tecnica di abbreviazione riduce la scrittura sulla tastiera e mostra chiaramente quello che sarà visualizzato sullo schermo.

Mantienilo semplice.

Se il tuo BASIC non permette questa abbreviazione, puoi usare una istruzione doppia sulla stessa linea:

```
50 PRINT "SCRIVI DUE INTERI";: INPUT ETA1, ETA2
```

•

La scelta dei nomi delle variabili

Dovresti scegliere sempre i nomi delle variabili in modo da poter facilmente ricordare che cosa significa il nome; altrimenti, ti potresti trovare in difficoltà a scrivere dei lunghi programmi e potresti facilmente cadere in errori accidentali. Inoltre, se i nomi delle variabili non sono chiari, in futuro potresti non essere più in grado di capire che cosa fa il programma.

Se il tuo BASIC permette solo l'uso di una lettera più un numero opzionale per i nomi di variabili, non c'è molto da fare. Ecco alcuni nomi significativi:

```
10 REM ORA USEREMO:
20 REM R   PER IL RISULTATO
30 REM P   PER IL PRIMO NUMERO
40 REM U   PER L'ULTIMO NUMERO
```

Se invece il tuo BASIC permette l'uso di nomi con lettere multiple, allora usali. Ecco l'esempio precedente, usando questa facoltà:

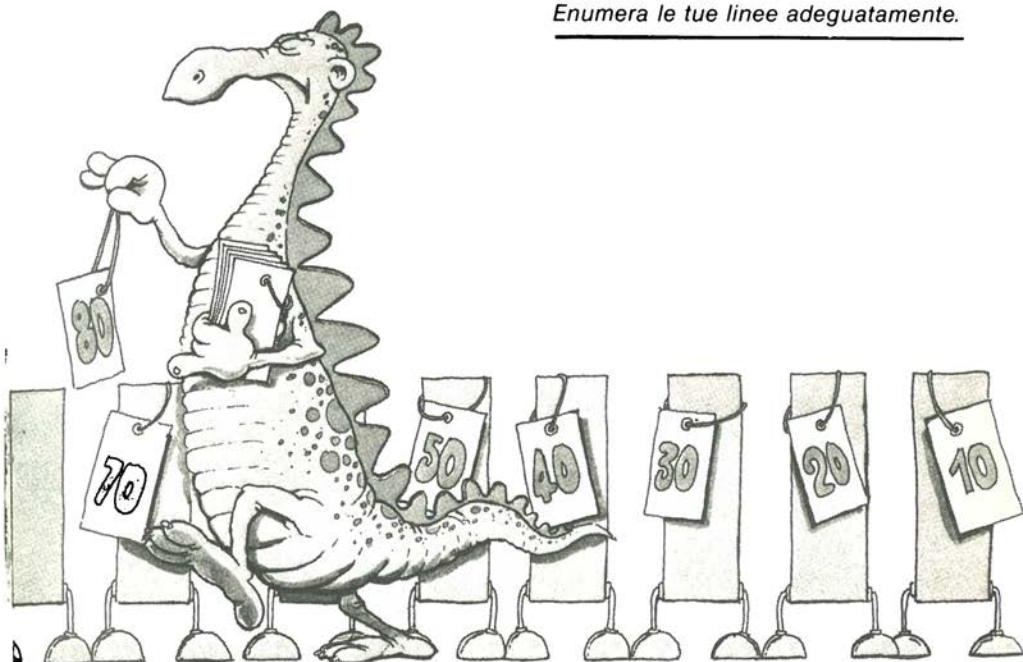
```
10 REM QUESTE SONO LE VARIABILI PER IL SECONDO NUMERO
20 REM      RISULTATO     PER IL RISULTATO
30 REM      PRIMO         PER IL PRIMO NUMERO
40 REM      ULTIMO        PER L'ULTIMO NUMERO
```

Esistono due tipi di restrizione:

- Il tuo BASIC pone normalmente un limite al numero dei caratteri.
- Non puoi usare come nome di variabile le parole riservate come PRINT, REM o INPUT.

È una buona idea elencare, all'inizio del programma, tutti i nomi delle variabili così come sono usati nelle formule o nelle equazioni.

Una adatta numerazione delle linee

Enumera le tue linee adeguatamente.

Finora in questo capitolo, abbiamo numerato le linee a multipli di 10:

10 (istruzione)

20 (istruzione)

30 (istruzione)

Tu puoi, comunque, usare qualsiasi sequenza; basta che usi i numeri interi positivi e non superi il limite imposto dal tuo interprete. Per esempio, puoi scrivere:

1 (istruzione)

2 (istruzione)

3 (istruzione)

oppure:

100 (istruzione)

200 (istruzione)

300 (istruzione)

Abbiamo preso la precauzione di lasciare un intervallo regolare tra due istruzioni consecutive in modo da poter successivamente aggiungere delle correzioni o dei miglioramenti. Per esempio, ecco la versione 1 di un programma:

```
10 REM * * * PROGRAMMA DI MOLTIPLICAZIONE * * *
20 INPUT "DAMMI 2 NUMERI"; N1, N2
30 PRINT "IL PRODOTTO E': "; N1 * N2
40 END
```

Ora, vogliamo rendere il programma più chiaro e migliorarne la visualizzazione. Per fare questo, scriveremo le seguenti istruzioni addizionali:

5 CLEAR

15 PRINT "QUESTO È UN PROGRAMMA DI MOLTIPLICAZIONE AUTOMATICA"

16 PRINT : PRINT : PRINT

35 PRINT : PRINT

Le nuove istruzioni verranno inserite automaticamente.
Listiamo ora il risultato:

```
>LIST
5 CLEAR
10 REM * * * PROGRAMMA DI MOLTIPLICAZIONE * *
15 PRINT "QUESTO E' UN PROGRAMMA
   DI MOLTIPLICAZIONE AUTOMATICA"
16 PRINT : PRINT : PRINT
20 INPUT "DAMMI 2 NUMERI"; N1,N2
30 PRINT "IL PRODOTTO E': "; N1 * N2
35 PRINT : PRINT
40 END
```

Dando il comando RUN alla nuova versione migliorata:

```
QUESTO E' UN PROGRAMMA
DI MOLTIPLICAZIONE AUTOMATICA
DAMMI 2 NUMERI? 12,15
IL PRODOTTO E': 180
```

Se prevedi di dover fare molte correzioni o aggiungere molte istruzioni, dovrai lasciare intervalli più ampi nella numerazione delle linee ed usare, per esempio:

10 (istruzione)

50 (istruzione)

60 (istruzione)

100 (istruzione)

Molti BASIC hanno un comando speciale chiamato RENUMBER che renumera tutte le linee del tuo programma ad intervallo di 10. Questo è molto utile nei programmi lunghi dove puoi uscire dalla numerazione per inserire dei pezzi o delle correzioni.

Riassunto

La scrittura di programmi che funzionino implica un ingrediente essenziale: la disciplina. È importante essere ordinati ed organizzati quanto più possibile nello scrivere programmi. Le abbreviazioni aumentano la possibilità d'errore. In particolare, il rendere chiari i tuoi programmi e le visualizzazioni ti richiederà tempo. In questo capitolo abbiamo descritto e sottolineato le tecniche necessarie per scrivere i programmi in modo chiaro.
Quando scrivi i tuoi programmi in BASIC, devi fare ogni sforzo per seguire i consigli dati in questo capitolo. È importante che tu prenda l'abitudine di una buona programmazione o altrimenti, i tuoi programmi potranno essere illeggibili e potranno non funzionare quando inizierai a scrivere dei programmi più complessi.

Esercizi

5.1: Descrivi le tecniche che migliorano la leggibilità della visualizzazione sullo schermo.

5.2: Sono permesse le seguenti istruzioni:

a. A = A + 1

b. A   =   A   +   1

c. PRINT   ALFA + 2

d. SOMMA=2 + (3 + (4/5))/2

e. IN   PUT NUMERO

f. SOMMA = 2 2 + 3 3

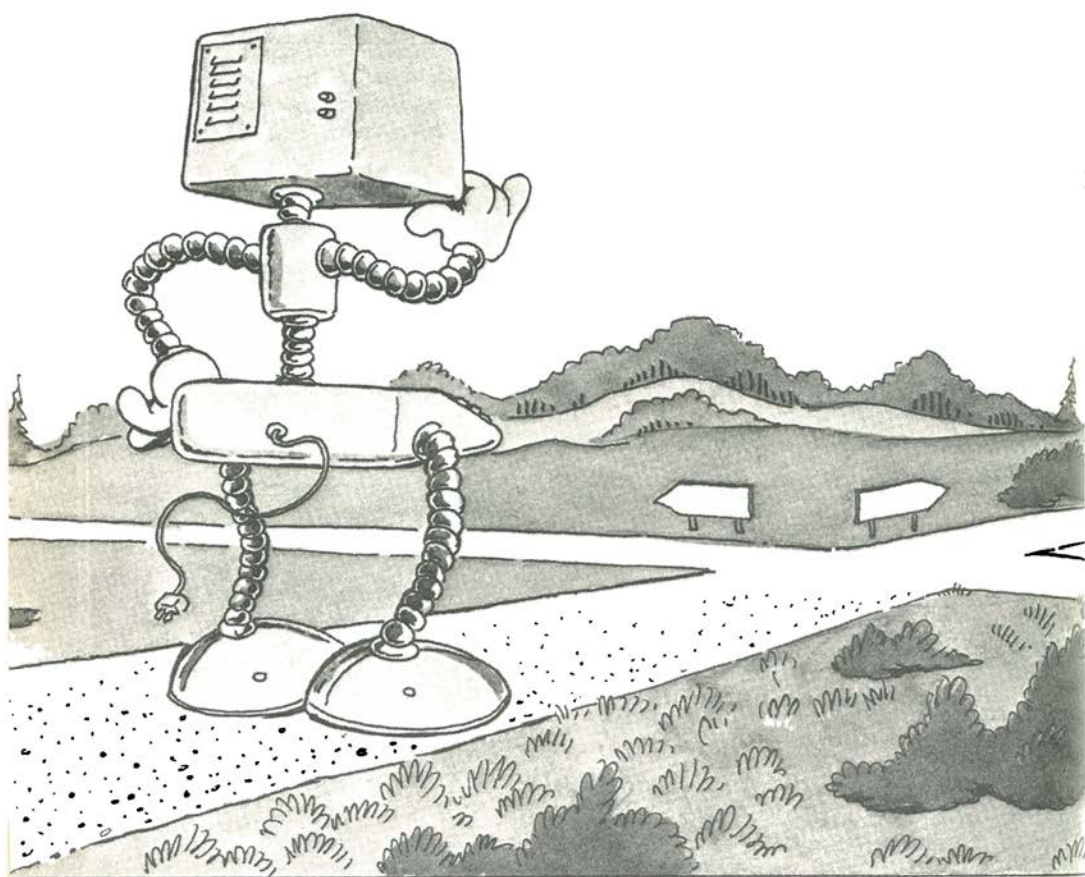
5.3: Spiega perchè la maggior parte degli INPUT dovrebbe avere un messaggio precedente.

5.4: Scrivi tre esempi di INPUT abbreviati.

5.5: Perchè usi le REM?

5.6: Quale è il valore di A dopo le due istruzioni seguenti:

```
30  A = 3
40  REM A = 4
```

Capitolo 6

Prendere le decisioni

Finora abbiamo imparato a comunicare con il computer e ad eseguire semplici operazioni aritmetiche. Comunque i nostri programmi sono stati un po' "stupidi" tanto che avremmo potuto ottenere facilmente gli stessi risultati a mano.
Questo perchè abbiamo utilizzato solamente le risorse elementari del computer e non abbiamo usufruito delle risorse più avanzate. Per esempio, i computer sono particolarmente bravi a compiere due cose: prendere decisioni (basate sulla logica e sui valori) ed eseguire compiti ripetitivi molte volte in poco tempo. Questo è ciò che impareremo a fare in questo e nel prossimo capitolo. In questo capitolo, in particolare, impareremo a prendere le decisioni. I nostri programmi in questo modo diventeranno intelligenti nel decidere cosa fare.
In BASIC, le decisioni in un programma vengono prese confrontando un valore tramite l'istruzione IF (se). Se il confronto è esatto viene eseguita una parte del programma; altrimenti, se il confronto è sbagliato ne viene eseguita un'altra. Impareremo ora come usare l'istruzione IF per eseguire dei test. Impareremo anche a usare l'istruzione GOTO (vai a) per eseguire un gruppo di istruzioni al di fuori della normale sequenza.

L'istruzione IF

L'istruzione IF è scritta così:

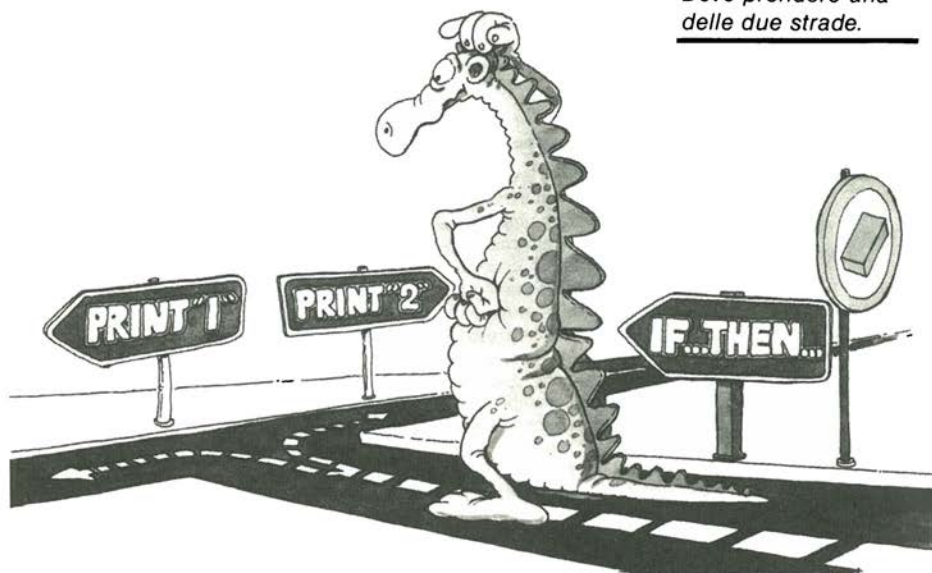
IF (condizione) THEN (espressione, cioè fa qualcosa)

Ecco un esempio:

IF I=1 THEN PRINT "UNO"

L'effetto di questa istruzione dovrebbe essere chiaro: se (IF) il valore della variabile è uguale ad 1 al momento in cui questa istruzione viene eseguita, in questo caso (THEN) visualizza la parola UNO. Se I non fosse uguale ad 1, non accadrebbe nulla e verrebbe eseguita l'istruzione successiva.
I=1 è chiamata *espressione logica*.
L'espressione I= 1 è *vera* quando I è uguale ad 1, in caso contrario è *falsa*.
L'istruzione IF...THEN ti permette di confrontare il valore di una espressione ed eseguire una istruzione o un'altra (cioè prendere una decisione) a seconda del risultato del test.
Ecco un altro esempio:

```
10 INPUT I
20 IF I = 1 THEN PRINT "UNO"
30 END
```

Osserva l'espressione imbarazzata di Dino. Deve prendere una delle due strade.

Ora esegui il programma con il RUN, premi 1 alla tastiera e lo schermo mostrerà:

```
>RUN
?1
UNO
>
```

Ora esegui di nuovo il programma, premi 2 alla tastiera e lo schermo sarà:

```
>RUN
?2
>
```

Questa volta, in risposta al 2, non è stato visualizzato alcun messaggio.
Insegnamo al nostro programma a riconoscere i numeri da 1 a 4:

```
10 REM QUESTO PROGRAMMA RICONOSCE I NUMERI DA 1 A 4
20 INPUT "SCRIVI UN NUMERO INTERO:"; NUMERO
30 IF NUMERO = 1 THEN PRINT "UNO"
40 IF NUMERO = 2 THEN PRINT "DUE"
50 IF NUMERO = 3 THEN PRINT "TRE"
60 IF NUMERO = 4 THEN PRINT "QUATTRO"
70 END
```

Mandiamo in esecuzione questo programma ed ecco visualizzato lo schermo dopo due esecuzioni:

```
>RUN
SCRIVI UN NUMERO INTERO:? 3
TRE
>RUN
SCRIVI UN NUMERO INTERO:? 5
>
```

Va bene, però non è perfetto; infatti, vorremmo che quando premiamo il 5, il programma risponda con una frase tipo:

NOŃ CONOSCO QUESTO NUMERO.

o con una nuova richiesta del numero.

C'è una opzione speciale dell'istruzione IF che può essere usata in questo caso. Per esempio, puoi scrivere:

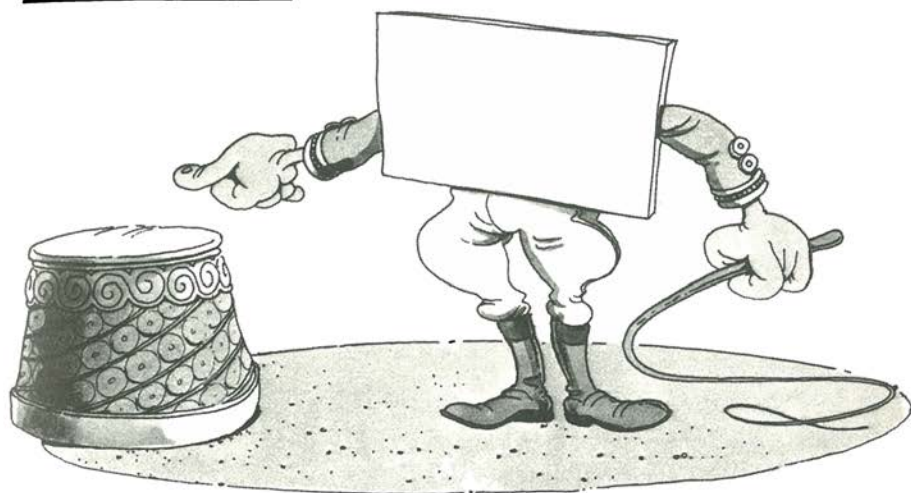
70 IF NUMERO=5 THEN 20

dove 20 è il numero della linea da eseguire se la condizione è vera. Questa è una forma nuova dell'istruzione IF. Questa istruzione significa che se NUMERO è uguale a 5 allora la prossima linea da eseguire è la numero 20. Ecco come possiamo saltare fuori dalla normale sequenza! Ecco un esempio:

```
10 INPUT I
20 IF I = 1 THEN 50
30 PRINT "NON HAI SCRITTO UN 1"
40 END
50 PRINT "HAI SCRITTO UN 1"
60 END
```

Esegui questo programma e scrivi 1 alla tastiera. Lo schermo diverrà così:

```
>RUN
?1
HAI SCRITTO UN 1
>
```

Possiamo far saltare il programma fuori dalla normale sequenza.

Ora esegui questo programma di nuovo e premi 2 alla tastiera. Ora il tuo schermo diverrà:

```
>RUN
?2
NON HAI SCRITTO UN 1
>
```

Il nostro programma è divenuto intelligente, cioè dà un messaggio appropriato a seconda che sia stato introdotto un 1 oppure no. Potrai chiederti se sia possibile ottenere lo stesso risultato usando la forma originale dell'istruzione IF. Proviamo:

```
10 INPUT I
20 IF I = 1 THEN PRINT "QUESTO E' UN UNO"
30 PRINT "QUESTO NON E' UN UNO"
40 END
```

Ora eseguiamo il programma e scriviamo 1 alla tastiera.
Lo schermo mostrerà:

Sono Bug. Ti ho preso.

```
>RUN
?1
QUESTO E' UN UNO
QUESTO NON E' UN UNO
```

Non va bene. Infatti, indipendentemente dall'esattezza o meno dell'IF, l'istruzione seguente nel programma (qui la linea numero 30), viene sempre eseguita.
In questo esempio, abbiamo ricevuto l'esatto messaggio quando è stata eseguita l'istruzione IF:

QUESTO È UN UNO

poi il secondo messaggio è stato visualizzato comunque:

QUESTO NON È UN UNO

La nuova forma dell'istruzione IF:

IF I=1 THEN 50

elimina questo problema. Useremo frequentemente questa istruzione nei nostri programmi.

Analizziamo ora più da vicino l'istruzione IF in modo da poterla usare in modo completo. La forma generale dell'istruzione IF...THEN è la seguente:

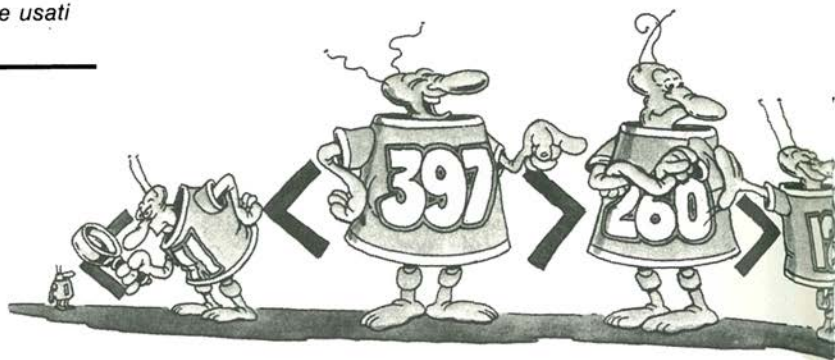
IF (espressione logica) THEN (espressione eseguibile o numero di linea).

Esaminiamo prima le espressioni logiche e poi le istruzioni eseguibili.

Un IF vero costringe l'interprete ad attivare l'istruzione.

Espressioni logiche

Nel nostro esempio, I=1 è una *espressione logica*, cioè può essere o *vera* o *falsa*. Vero e falso sono chiamati *valori logici*. Ecco alcuni esempi di espressioni logiche:

Gli operatori relazionali possono essere usati sulle variabili.

I=1 (*I uguale a 1*)

I>4 (*I maggiore di 4*)

NUMERO<100 (*NUMERO minore di 100*)

ANNO<>5 (*ANNO diverso da 5*)

ETA<13 (*ETA minore di 13*)

Una espressione logica combina valori e variabili con operatori logici. Per completezza, gli operatori logici che puoi usare sono i seguenti:

= uguale

<> disuguale (*in matematica si scrive ≠ o*)

< minore

> maggiore

<= minore o uguale (*in matematica si scrive* ⩽)

>= maggiore o uguale (*in matematica si scrive* ⩾)

Ecco alcune espressioni logiche più complesse:

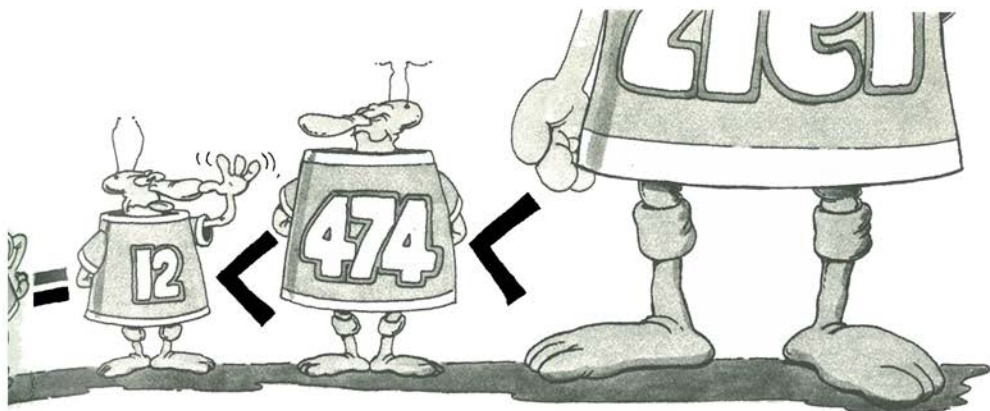
(NUMERO + 2) > 4

(ETA — 5) >= 10

((2 * I — 5)/2) < 10

2 > I

ma puoi anche scrivere:

4 > 2 (*questa è sempre vera*)

4 = 2 *(questa è sempre falsa)*

Le seguenti espressioni, invece non sono permesse:

2 < I < *0 (può essere usato solo un operatore relazionale)*

(2ETA — 2) < 5 *(espressione errata, manca un *. Dovrebbe essere scritta: (2*ETA – 2)<5)*

Puoi anche combinare le espressioni logiche usando gli *operatori logici* AND, OR, e NOT. per esempio, puoi scrivere:

IF (ETA>9) AND (ETA<20) THEN PRINT "QUESTO È UN TEENAGER"

Questa istruzione visualizza "QUESTO È UN TEENAGER" quando ETA è maggiore di 9 e minore di 20. L'AND è soddisfatta (cioè è vera) quando entrambe le clausole sono vere. Nota che non puoi scrivere:

IF ETA (>9 AND <20) THEN ...

perchè dentro le parentesi deve esserci una espressione valida.
Ecco un altro esempio che usa due espressioni logiche:

```
20 INPUT RISPOSTA$
30 IF (RISPOSTA$ = "SI") OR (RISPOSTA$ = "NO") THEN 60
40 PRINT "RISPOSTA NON VALIDA"
   ...
60 PRINT "RISPOSTA VALIDA - ANDIAMO AVANTI"
```

Questo programma memorizza una risposta nella variabile RISPOSTA$. (Ricorda che la $ alla fine di un nome è usata per indicare che si tratta di una variabile stringa, cioè un insieme di caratteri). SI e NO sono le uniche risposte valide e questo segmento di programma controlla la validità di ciò che hai scritto.

Se scrivi SI, allora (RISPOSTA$ = "SI") è vera, l'IF ha successo e perciò viene eseguita successivamente la linea 60 che visualizza:

RISPOSTA VALIDA - ANDIAMO AVANTI

Ugualmente se la risposta è NO.
Se invece scriverai qualcosa di diverso, riceverai il messaggio:

RISPOSTA NON VALIDA

Un OR è soddisfatto, cioè *vero*, quando almeno una delle due clausole è vera. L'OR non è soddisfatto quando entrambe le clausole sono false.
Per esempio, se scrivi "FORSE", allora (RISPOSTA$="SI") è falsa e viene tentata la seconda clausola (RISPOSTA$ ="NO"). Anch'essa è falsa e viene visualizzato il messaggio:

RISPOSTA NON VALIDA

Per finire, NOT può essere usato per negare una condizione: ecco un esempio di una complessa istruzione IF:

IF((MEDIA<*5.5)AND(ULTIMOESAME<5.0)AND NOT(ORALE>6.0))THEN PRINT "BOCCIATO"*

In questa istruzione, abbiamo valutato tre condizioni.
Non discuteremo più a lungo questa espressione, ma puoi farci da solo degli esperimenti.

Istruzioni eseguibili

Ricordiamo la definizione dell'istruzione IF:

IF (espressione logica) THEN (istruzione eseguibile o numero di linea)
Ora che abbiamo fatto pratica con le espressioni logiche, esaminiamo la parte destra di questa definizione:

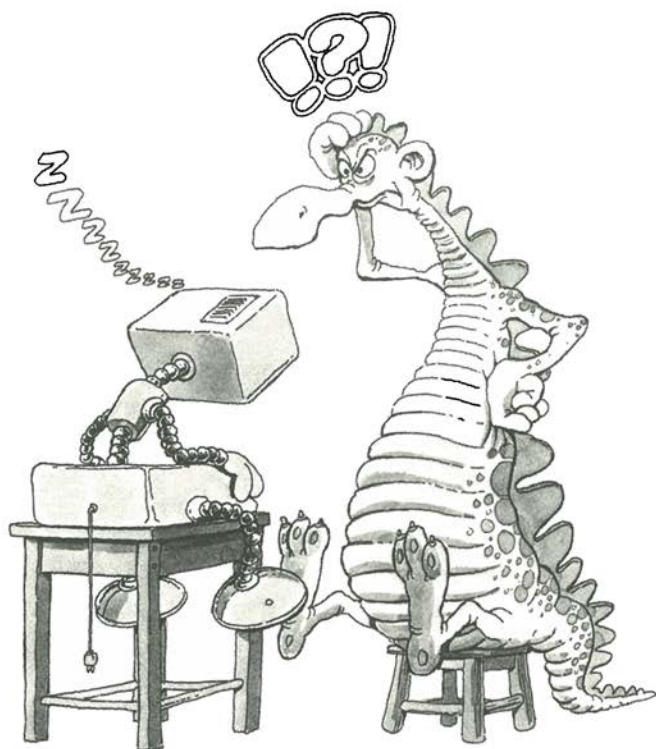
THEN (istruzione eseguibile o numero di linea)

Una istruzione eseguibile è semplicemente una istruzione che viene eseguita, essa può essere un'assegnazione, una istruzione INPUT o PRINT. Non può essere invece un'altra istruzione IF o un comando come REM, CLEAR, NEW o LIST.
Ora che abbiamo capito la teoria dell'istruzione IF, mettiamola in pratica.

Useremo le nostre conoscenze per sviluppare un programma che mostri un "menu" sullo schermo. A seconda della scelta fatta dall'utente, questo programma didattico eseguirà addizioni, sottrazioni, moltiplicazioni o divisioni.
Ecco il dialogo che prevediamo di generare sullo schermo:

```
BENVENUTO DALL'INSEGNANTE COMPUTERIZZATO
VALUTERO' LA TUA BRAVURA IN ARITMETICA
IN CHE COSA VUOI FAR PRATICA ?
  - ADDIZIONE        (SCRIVI 1)
  - SOTTRAZIONE      (SCRIVI 2)
  - MOLTIPLICAZIONE  (SCRIVI 3)
  - DIVISIONE        (SCRIVI 4)
QUALE SCEGLI:? 3

VA BENE. MOLTIPLICHIAMO.
QUANTO FA 2 PER 3: 6
VA BENE. CONGRATULAZIONI.
```

"Posso proporti un menu aritmetico?".

Ecco il listato del programma che lo realizza:

```
10  REM QUESTO PROGRAMMA TI ESERCITA IN ARITMETICA
20  PRINT "BENVENUTO DALL'INSEGNANTE COMPUTERIZZATO"
30  PRINT "VALUTERO' LA TUA BRAVURA IN ARITMETICA"
40  PRINT "IN CHE COSA VUOI FARE PRATICA?"
50  PRINT " - ADDIZIONE           (SCRIVI 1)"
60  PRINT " - SOTTRAZIONE         (SCRIVI 2)"
70  PRINT " - MOLTIPLICAZIONE     (SCRIVI 3)"
80  PRINT " - DIVISIONE           (SCRIVI 4)
90  INPUT "QUALE SCEGLI:"; SCELTA
100 IF (SCELTA = 1) THEN 200
110 IF (SCELTA = 2) THEN 300
120 IF (SCELTA = 3) THEN 400
130 IF (SCELTA = 4) THEN 500
140 PRINT "SCELTA NON CORRETTA. DEVI SCEGLIERE UN NUMERO
    TRA 1 E 4"
150 PRINT "CIAO" : END
190 REM - - - - - ADDIZIONE - - - - -
200 PRINT "VA BENE. ADDIZIONIAMO"
210 INPUT "QUANTO FA 4 + 7 :"; NUMERO
220 IF (NUMERO <> 11) THEN 600
230 PRINT "VA BENE. CONGRATULAZIONI" : END
290 REM - - - - - SOTTRAZIONE - - - - -
300 PRINT "VA BENE. SOTTRAIAMO"
310 INPUT "QUANTO FA 9 - 5 :"; NUMERO
320 IF (NUMERO <> 4) THEN 600
330 PRINT "VA BENE. CONGRATULAZIONI" : END
390 REM - - - - - MOLTIPLICAZIONE - - - - -
400 PRINT "VA BENE. MOLTIPLICHIAMO"
410 INPUT "QUANTO FA 2 PER 3 :"; NUMERO
420 IF (NUMERO <> 6) THEN 600
430 PRINT "VA BENE. CONGRATULAZIONI" : END
490 REM - - - - - DIVISIONE - - - - -
500 PRINT "VA BENE. DIVIDIAMO"
510 INPUT "QUANTO FA 9 DIVISO 3 :"; NUMERO
520 IF (NUMERO <> 3) THEN 600
530 PRINT "VA BENE. CONGRATULAZIONI" : END
590 REM - - - - - USCITA PER ERRORE - - - - -
600 PRINT "SBAGLIATO. MI SPIACE. CIAO." : END
```

nelle istruzioni IF non sono richieste, ma migliorano la leggibilità). Se l'utente scrive "1", allora (SCELTA=1) è vera e viene eseguita successivamente la linea 200. Se l'utente scrive qualcosa di diverso da 1, 2, 3 o 4, allora viene eseguita l'istruzione 140 ed il programma avverte:

Questo programma sembra imponente, ma in pratica è molto semplice. Esaminiamolo. Le istruzioni da 20 a 90 producono la visualizzazione o "menu" sullo schermo. Il programma nelle istruzioni da 100 a 130 chiede all'utente di fare una scelta (le parentesi e si ferma (questa è l'istruzione: END alla linea 150).

```
SCELTA NON CORRETTA. DEVI SCEGLIERE
UN NUMERO
TRA 1 E 4
CIAO
>
```

"Congratulazioni!"

Nel nostro esempio, scriviamo 3. L'istruzione 100 non ha successo ed allora viene eseguita l'istruzione 110. Anche essa non è vera, perciò viene eseguita l'istruzione 120. L'istruzione 120 è vera, perchè (SCELTA=3) è vera e quindi viene eseguita successivamente l'istruzione 400. Ecco il corrispondente segmento di programma:

```
400 PRINT "VA BENE. MOLTIPLICHIAMO"
410 INPUT "QUANTO FA 2 PER 3 :"; NUMERO
420 IF (NUMERO <> 6) THEN 600
430 PRINT "VA BENE. CONGRATULAZIONI" : END
```

Nel nostro esempio, scriviamo 6 in risposta all'istruzione 410. Quando l'istruzione 420 viene eseguita, (NUMERO<>6) è falsa perchè NUMERO=6. (Ricorda che <> significa non uguale).
Allora l'istruzione successiva è la 430 ed il programma risponde con:

e si ferma perchè la linea 430 contiene *due* istruzioni, la seconda delle quali è:

: END

Osservando questo programma, puoi subito avere una nuova delusione: se scrivi un numero diverso da 1,2,3 o 4 dopo il menu o se dai una risposta errata di aritmetica, il programma si ferma bruscamente. Idealmente, noi vorremmo che il programma continuasse. Per esempio, sarebbe una buona idea che il programma, dopo aver avvertito l'utente che un numero diverso da 1 a 4 non è permesso, richieda una nuova scelta. Vorremmo poter ritornare all'inizio del programma e ripartire daccapo o più in generale poter saltare in qualunque parte del programma. Questo è possibile tramite l'istruzione GOTO. Esaminiamola dettagliatamente.

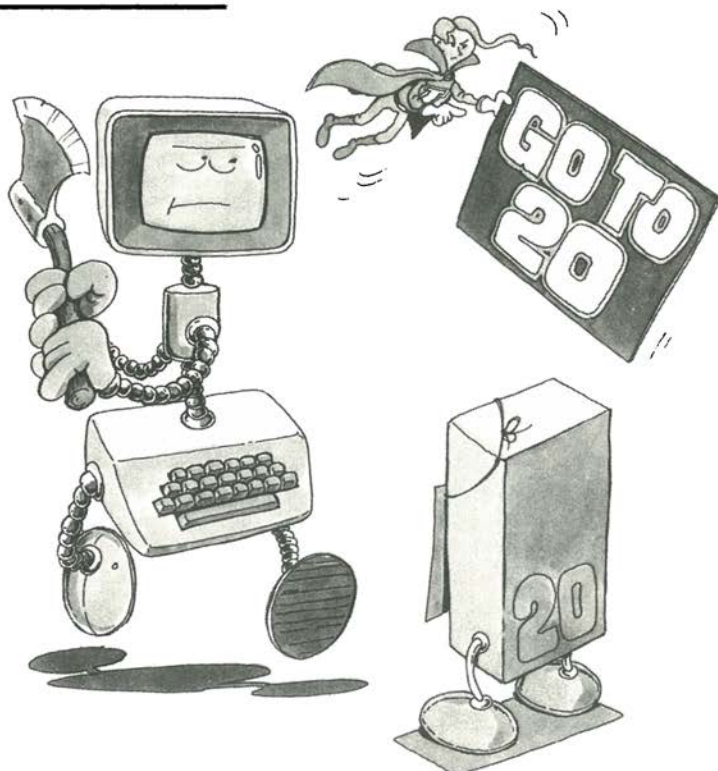
L'istruzione GOTO

L'istruzione GOTO viene scritta:

GOTO (numero linea)

Essa rinvia l'esecuzione alla linea specificata. Eccone un esempio:

```
10 PRINT "QUESTO PROGRAMMA RICONOSCE GLI 1.
   SCRIVI 0 PER SMETTERE."
20 INPUT "SCRIVI UN NUMERO:"; NUMERO
30 IF NUMERO = 1 THEN PRINT "UNO"
40 IF NUMERO = 0 THEN 60
50 GOTO 20
60 END
```

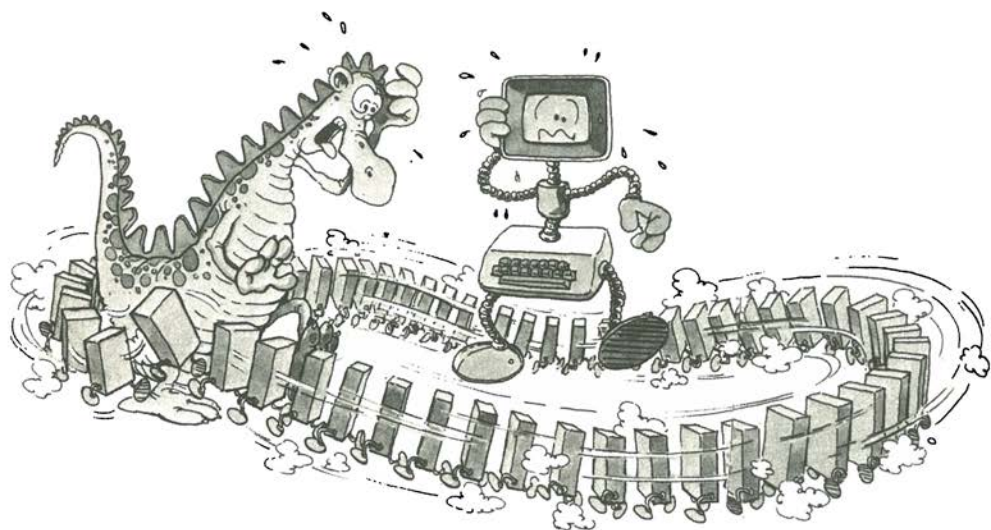
GOTO costringe l'esecuzione di specifiche istruzioni.

Ecco un esempio di esecuzione:

```
>RUN
QUESTO PROGRAMMA RICONOSCE GLI 1.
SCRIVI 0 PER SMETTERE.
SCRIVI UN NUMERO:? 1
UNO
SCRIVI UN NUMERO:? 5
SCRIVI UN NUMERO:? 25
SCRIVI UN NUMERO:? 1
UNO
SCRIVI UN NUMERO:? 0
>
```

Ogni volta che scrivi 1 il programma lo riconosce e scrive UNO. Ogni volta che scrivi ogni altro valore, il numero è ignorato e il programma richiede un altro valore.
Il programma ritorna continuamente all'inizio. Questo è chiamato *ciclo o loop*. Se scrivi uno 0, il programma lo riconosce, salta alla linea 60 e termina l'esecuzione.
Togliamo ora l'istruzione 40. Il programma diverrà:

```
10 PRINT "QUESTO PROGRAMMA RICONOSCE GLI 1.
   SCRIVI 0 PER SMETTERE."
20 INPUT "SCRIVI UN NUMERO:"; NUMERO
30 IF NUMERO = 1 THEN PRINT "UNO"
50 GOTO 20
60 END
```

Ecco l'esecuzione:

```
QUESTO PROGRAMMA RICONOSCE GLI 1.
SCRIVI 0 PER SMETTERE.
SCRIVI UN NUMERO:? 2
SCRIVI UN NUMERO:? 1
UNO
SCRIVI UN NUMERO:? 5
SCRIVI UN NUMERO:? 0
SCRIVI UN NUMERO:?
```

"Ferma questo ciclo!"

Come l'apprendista stregone, abbiamo creato un terribile problema: il programma non si arresta più! Questo è un errore frequente di programmazione chiamato *ciclo senza fine*. L'esecuzione del programma può continuare in eterno. Non ti preoccupare, non danneggerai nulla. Per fermarlo, devi premere il tasto previsto dall'interprete per arrestare un programma BASIC (prova CTRL C). Se le cose volgono al peggio e non riesci a ricordare cosa fare, spegni il tuo computer e poi riaccendilo. Ma ricordati: se spegni il computer, perderai tutto quello che hai scritto finora e che non hai precedentemente memorizzato su una cassetta o un dischetto. Ci sforzeremo di evitare queste situazioni spiacevoli prevedendo, da ora in poi, per ogni programma, una normale (programmata) uscita.
Avendo introdotto l'istruzione GOTO, ritorniamo alla nostra definizione di IF per semplificarlo.

L'istruzione IF rivisitata

Ricorda che una forma dell'istruzione IF era:

IF (espressione logica) THEN (numero linea)

e l'altra:

IF (espressione logica) THEN (istruzione eseguibile).

Ecco un esempio della prima forma:

IF NUMERO=0 THEN 60

che è equivalente a:

IF NUMERO=0 THEN GOTO 60

GOTO 60 è una istruzione eseguibile e puoi renderti conto che la forma

THEN 60

è una comoda abbreviazione di

THEN GOTO 60 .

Così, in realtà, la forma generale di una istruzione IF è, in effetti, più semplice della nostra precedente definizione:

IF (espressione logica) THEN (istruzione eseguibile).

C'è, inoltre, una ulteriore semplificazione: l'uso di THEN davanti a GOTO è generalmen-

te opzionale. Per la maggior parte dei BASIC, le seguenti espressioni sono, perciò, equivalenti:

```
IF NUMERO=0 THEN 100
IF NUMERO=0 THEN GOTO 100
IF NUMERO=0 GOTO 100
```

Mostreremo ora l'uso delle istruzioni IF e GOTO in alcuni esempi.

Parliamo di nuovo dei contatori

Nel capitolo 5, abbiamo introdotto la tecnica del contatore. Usiamola per contare il numero di 1 scritti nell'ultimo programma del precedente paragrafo:

```
1   REM CONTATORE DI UNO
10  PRINT "CONTERO' QUANTI 1 SCRIVERAI"
20  PRINT "SCRIVI 0 PER SMETTERE"
30  SOMMA = 0
40  INPUT "SCRIVI UN NUMERO: "; NUMERO
50  IF NUMERO = 0 THEN 100
60  IF NUMERO <> 1 THEN GOTO 40
70  SOMMA = SOMMA + 1
80  PRINT "UNO. TOTALE FINORA: "; SOMMA
90  GOTO 40
100 END
```

Ecco l'esecuzione sullo schermo:

```
CONTERO' QUANTI 1 SCRIVERAI.
SCRIVI 0 PER SMETTERE
SCRIVI UN NUMERO:? 10
SCRIVI UN NUMERO:? 1
UNO. TOTALE FINORA: 1
SCRIVI UN NUMERO:? 9
SCRIVI UN NUMERO:? 5
SCRIVI UN NUMERO:? 1
UNO. TOTALE FINORA: 2
SCRIVI UN NUMERO:? 2
SCRIVI UN NUMERO:? 1
UNO. TOTALE FINORA: 3
SCRIVI UN NUMERO:? 410
SCRIVI UN NUMERO:?
```

Esaminiamo il programma: Le istruzioni 10 e 20 visualizzano messaggi:

```
10   PRINT "CONTERO' QUANTI 1 SCRIVERAI"
20   PRINT "SCRIVI 0 PER SMETTERE"
```

L'istruzione 30 inizializza il contatore SOMMA a zero:

```
30   SOMMA=0
```

Poi il numero è richiesto dalla tastiera:

```
40   INPUT "SCRIVI UN NUMERO: ";NUMERO
```

Se il numero è 0, abbiamo

```
50   IF NUMERO=0 THEN 100
```

dove 100 è una istruzione END. Assumiamo che il numero sia 10 è vediamo cosa accade:

```
60   IF NUMERO<>1 THEN GOTO 40
```

cioè se il numero è diverso da 1, si salta alla linea 40 che è di nuovo la richiesta di un numero. Se il numero è 1, si va avanti:

70 SOMMA=SOMMA+1

Il contatore SOMMA viene incrementato di uno. Ricorda il significato di una istruzione di assegnazione: puoi leggere la linea 70 come:

SOMMA riceve il valore di (vecchio valore di SOMMA) + 1
A questo punto, SOMMA riceve il valore 0+1=1. La prossima istruzione è:

80 PRINT "UNO. TOTALE FINORA: "; SOMMA

Poi il programma ritorna di nuovo alla linea 40 per richiedere un nuovo numero:

90 GOTO 40

Esercizio di aritmetica rivisitato

Ricordi che abbiamo sviluppato un programma per esercitarsi in aritmetica all'inizio di questo capitolo? Ci rammaricavamo del fatto che era troppo semplice e non potevamo riciclarlo. Ora lo possiamo.
Poichè il programma era abbastanza lungo, rivediamo solo il segmento interessato. Ecco come appariva, prima, il segmento riguardante la scelta di un numero tra 1 e 4:

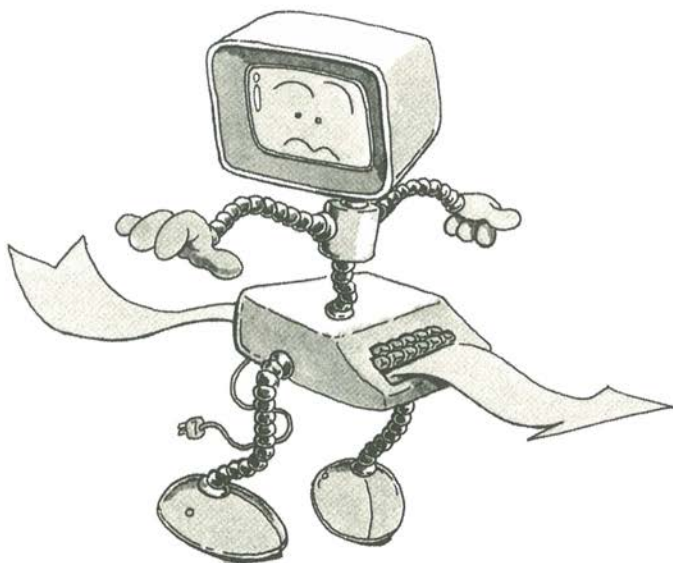
```
90  INPUT "QUALE SCEGLI:"; SCELTA
100 IF (SCELTA = 1) THEN 200
110 IF (SCELTA = 2) THEN 300
120 IF (SCELTA = 3) THEN 400
130 IF (SCELTA = 4) THEN 500
140 PRINT "SCELTA NON CORRETTA.
    DEVI SCEGLIERE UN NUMERO TRA 1 E 4"
150 PRINT "CIAO" : END
```

ed ecco il nostro miglioramento:

```
150 GOTO 90
```

Questo è tutto. Provalo.
Ora ci piacerebbe che il programma presentasse più di un solo esercizio; per esempio, dieci esercizi diversi. Potremmo farlo aggiungendo dei GOTO ed un contatore.

Controllare i dati in ingresso

Gli esempi che abbiamo fatto mostrano una importante regola per progettare dei programmi: qualunque siano i dati richiesti alla tastiera, non ritenere mai che essi siano stati introdotti sicuramente in modo corretto. Un utente potrebbe sbagliare tasto deliberatamente o accidentalmente.
Per evitare comportamenti strani o sbagliati, *controlla sempre i dati in ingresso*, cioè se l'informazione scritta alla tastiera non è esatta, genera un messaggio e richiedi il dato di nuovo.

"Controlla bene i dati nel mio ingresso, per piacere".

Eseguiremo questo controllo dei dati di ingresso nella maggior parte nei nostri esempi. Sviluppiamo ora due programmi completi che prendano delle decisioni.

Conversione miglia-chilometri

Nel capitolo 3, abbiamo imparato ad eseguire una semplice conversione da miglia a chilometri. Rendiamola automatica:

```
10    REM * * * CONVERSIONE MIGLIA-CHILOMETRI * * *
20    REM
30    PRINT "CONVERTO MIGLIA IN CHILOMETRI"
40    PRINT "SCRIVI 0 PER SMETTERE"
50    INPUT "QUANTE MIGLIA "; MIGLIA
60    IF MIGLIA = 0 THEN GOTO 100
70    KM = MIGLIA * 1.6
80    PRINT MIGLIA; " MIGLIA = "; KM; " CHILOMETRI"
90    GOTO 50
100   END
```

Ecco l'esecuzione:

```
CONVERTO MIGLIA IN CHILOMETRI
SCRIVI 0 PER SMETTERE
QUANTE MIGLIA? 7
7 MIGLIA = 11.2 CHILOMETRI
QUANTE MIGLIA? 10
10 MIGLIA = 16 CHILOMETRI
QUANTE MIGLIA? 0
```

Compleanno

Ecco un altro esempio. Miglioriamo uno dei nostri primi programmi che calcolava l'età di una persona. Immetteremo ora la data odierna e l'anno, il mese ed il giorno in cui sei nato ed il programma ci darà la tua età esatta. Ecco il programma:

```
10   REM * * * CALCOLO DELL'ETA' * * *
20   INPUT "QUALE E' IL TUO NOME"; NOME$
30   PRINT "SALVE "; NOME$; " CALCOLERO' LA TUA ETA'"
40   PRINT "CHE GIORNO E' OGGI? (AAMMGG)"
50   INPUT "PRIMA L'ANNO (2 CIFRE): "; AA
60   IF (A0 OR A > 99) THEN 50
70   INPUT "IL MESE (DA 1 A 12) : "; MM
80   IF (MM < 1 OR MM > 12) THEN 70
90   INPUT "IL GIORNO: "; GG
100  IF (GG < 1 OR GG > 31) THEN 90
110                                                REM
120  PRINT "ORA DAMMI LA TUA DATA DI NASCITA"
130  INPUT "ANNO (2 CIFRE) : "; ANASC
140  IF (ANASC < 0 OR ANASC > 99) THEN 130
150  INPUT "MESE (DA 1 A 12) : "; MNASC
160  IF (MNASC < 1 OR MNASC > 12) THEN 150
170  INPUT "GIORNO: "; GNASC
180  IF (GNASC < 1 OR GNASC > 31) THEN 170
190                                                REM
200  REM - - - - - CALCOLO DELL'ETA' - - - - -
210  IF MNASC < MM THEN 270
220  IF MNASC > MM THEN 320
230  REM - - - - - COMPLEANNO IN QUESTO MESE - - - - -
240  IF GNASC < GG THEN 270
250  IF GNASC > GG THEN 320
260  PRINT "OGGI E' IL TUO COMPLEANNO. CONGRATULAZIONI"
270  ETA = AA - ANASC
280  PRINT "TU HAI "; ETA; " ANNI"
290  END
300                                                REM
310  REM COMPLEANNO NON ANCORA COMPIUTO IN QUEST'ANNO
320  ETA = AA - ANASC - 1
330  GOTO 280
340  END
```

Nota: Se il vostro interprete BASIC non accetta parole chiave all'interno di nomi di variabile, dovete sostituire, come appare nel listato precedente, le variabili del tipo DATA NASC (DATA è una parola chiave) con ANASC.

Nonostante la sua lunghezza, il programma è molto semplice. Osserva bene come abbiamo controllato ogni dato di ingresso, anche se per non rendere il programma troppo lungo, il controllo è approssimato. Non abbiamo controllato che ogni numero fosse intero, nè il numero dei giorni di ogni mese. Questo è lasciato come esercizio al lettore meticoloso (e paziente).
Ecco il modo in cui potresti controllare che MM sia un numero intero compreso tra 1 e 12:

```
IF NOT (MM=1 OR MM=2 ..... OR MM=12) THEN 70
```

Riassunto

Usando le istruzioni IF e GOTO, abbiamo imparato a scrivere programmi che eseguano dei confronti sui valori e che prendano decisioni. Abbiamo anche imparato ad eseguire dei cicli in modo che una parte di un programma possa essere ripetuta indefinitamente. Inoltre abbiamo imparato a controllare sistematicamente i dati introdotti dalla tastiera. Ora possediamo tutte le conoscenze pratiche richieste per scrivere i programmi comuni ed abbiamo esaminato anche parecchi esempi significativi. In seguito semplificheremo la scrittura di un programma.
A causa della frequenza dei cicli e delle automazioni in ogni programma, il BASIC offre delle facilitazioni addizionali sotto forma di altre istruzioni. Parleremo di queste agevolazioni nel prossimo capitolo.

6.1: Qual è l'uso dell'istruzione IF?

6.2: Qual è il risultato del seguente programma:

```
10 INPUT RISPOSTA$
20 IF (RISPOSTA$ = "SI") THEN PRINT "GRAZIE"
30 IF (RISPOSTA$ = "NO") THEN PRINT "TROPPO SCORTESE"
40 PRINT "SI O NO":GOTO 10
```

Se il risultato non è logico, suggerisci un programma migliore.

6.3: Sono valide le seguenti espressioni logiche?

a. A = 4

b. B = 2 OR C = 3

c. A > 5

d. 5 > A

e. 1 > 2

f. SOMMA > NUMERO

g. LETTERA$ = "A"

6.4: È valido quanto segue?

10 IF A=5 THEN IF B=2 THEN 18

6.5: Cosa è un ciclo in un programma?

Capitolo 7

Le ripetizioni automatiche

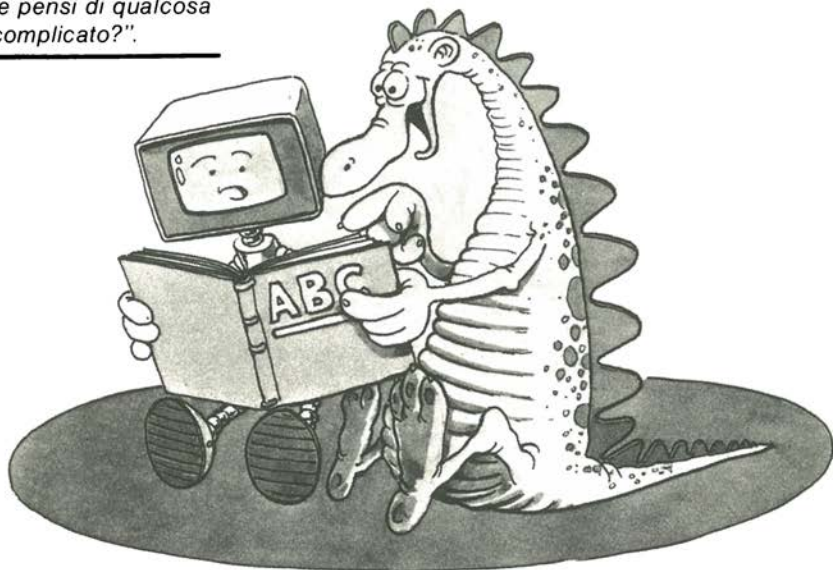
Usando le istruzioni IF e GOTO, possiamo eseguire ripetutamente alcune parti di un programma. I corrispondenti segmenti sono detti cicli o loop e la maggior parte dei programmi usano i cicli. In questo capitolo impareremo tecniche migliori per la creazione di un loop. Svilupperemo anche dei programmi più sofisticati per automatizzare un lavoro. Inizieremo questo capitolo rivedendo le tecniche di IF/GOTO per la generazione di un loop. Poi introdurremo una nuova istruzione, l'istruzione FOR-...NEXT, per facilitare la creazione dei cicli ed useremo estesamente questa importante istruzione nei nostri programmi.

La tecnica IF/GOTO

Inizieremo esaminando un programma che automatizza un ciclo, usando la tecnica IF/GOTO. Esaminando questo programma, puntualizzeremo su certe caratteristiche comuni a tutti i loop. Per esempio, esamineremo l'uso del contatore, l'incremento, l'inizializzazione e il confronto prima dell'uscita dal ciclo. Ecco il programma che calcola la somma dei primi dieci numeri interi.

```
1  REM  * * * SOMMA DEI PRIMI 10 NUMERI INTERI * * *
10 SOMMA = 0
20 I = 1
30 SOMMA = SOMMA + I
40 I = I + 1
50 IF I = 11 THEN 70
60 GOTO 30
70 PRINT "LA SOMMA DEI PRIMI 10 NUMERI INTERI E': ";
   SOMMA
80 END
```

In questo programma sono usate due variabili: SOMMA ed I: la variabile SOMMA accumula la somma dei primi 10 interi durante l'addizione (è equivalente al totale parziale di una calcolatrice). I è il numero intero che viene sommato a SOMMA.

"Che ne pensi di qualcosa di più complicato?".

Ricordati che ad una variabile deve essere assegnato un valore la prima volta che viene usata. Perciò, prima di usare SOMMA ed I in una formula, dobbiamo assegnare loro un valore *iniziale* (rispettivamente 0 e 1). Ciò è stato fatto con le istruzioni 10 e 20. Queste istruzioni vengono dette istruzioni di *inizializzazione*.
L'istruzione seguente è

30 SOMMA = SOMMA + I

"Ti ricordi di me?".

Questa istruzione aggiunge il valore corrente di I al valore corrente di SOMMA. Quando questa istruzione viene eseguita la prima volta, il valore di SOMMA è 0 ed il valore di I è 1. Ne risulta che questa istruzione assegna il valore 0+1=1 a SOMMA. Dopo l'esecuzione di questa istruzione, SOMMA contiene il valore 1.
L'istruzione seguente è:

40 I = I + 1

Il valore corrente di I è 1. Il risultato di questa istruzione è di dare ad I il nuovo valore 2. Questa è la tecnica del contatore: I è incrementato di uno per generare il prossimo numero intero. Allo stesso tempo, I indica anche quanti numeri interi sono stati addizionati finora. In altre parole la variabile I è usata sia come corrente numero intero che come contatore.
A questo punto, ciò che rimarrebbe da fare sarebbe di tornare all'istruzione 30 e continuare ad aggiungere numeri interi:

50 GOTO 30

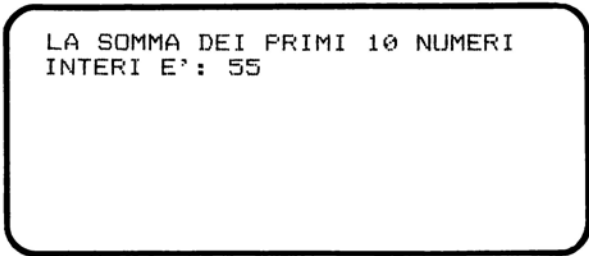
"Ti ho preso di nuovo!".

Sbagliato. Questo programma (in teoria) non si fermerà mai (in pratica si arresterà non appena che SOMMA raggiungerà il massimo valore permesso dal tuo interprete).
Non è questo quello che volevamo. Noi volevamo che il programma si arrestasse dopo avere eseguito il loop 10 volte. Dobbiamo perciò introdurre una istruzione di controllo. Eccola:

50 IF I = 11 THEN 70

Una volta che I raggiunge il valore 11, viene eseguita l'istruzione 70 ed il programma si arresta. Questa fase è chiamata *uscita* (exit) dal ciclo. Verifichiamo che il valore 11 (piuttosto che 10) è corretto nella istruzione 10.
Se scrivessimo:

```
50  IF I=10 THEN 70
```

non funzionerebbe, perchè quando I raggiunge il valore 10, SOMMA contiene solamente la SOMMA da 1 a 9. Il ciclo deve essere eseguito quindi ancora una volta.
Ricordati che un ciclo contiene sempre un contatore e dovrai sempre controllare attentamente il valore del contatore che provoca l'uscita dal loop. Nel nostro esempio, finchè I non è uguale a 11, il ciclo viene rieseguito:

```
60  GOTO 30
```

Non appena I raggiunge il valore 11, i primi dieci numeri interi sono sommati. Ciò accade perchè nel nostro programma, l'addizione (SOMMA = SOMMA + I) ha luogo prima dell'incrementazione (I = I + 1). Le due istruzioni finali del programma determinano l'*uscita* dal ciclo:

```
70  PRINT "LA SOMMA DEI PRIMI 10 NUMERI INTERI E': "; SOMMA
80  END
```

Ecco l'esecuzione:

L'illustrazione in Figura 7.1 mostra il flusso di controllo del programma. I numeri tra parentesi sono i numeri delle istruzioni.
Questo diagramma è detto *flowchart* o *diagramma di flusso*. Tratteremo dettagliatamente questo argomento nel Capitolo 8. Per ora, osserva semplicemente l'organizzazione generale del programma: inizializzazione, calcolo ed incrementazione, confronto e uscita. Tutti i segmenti di programma che hanno un ciclo eseguono queste funzioni.

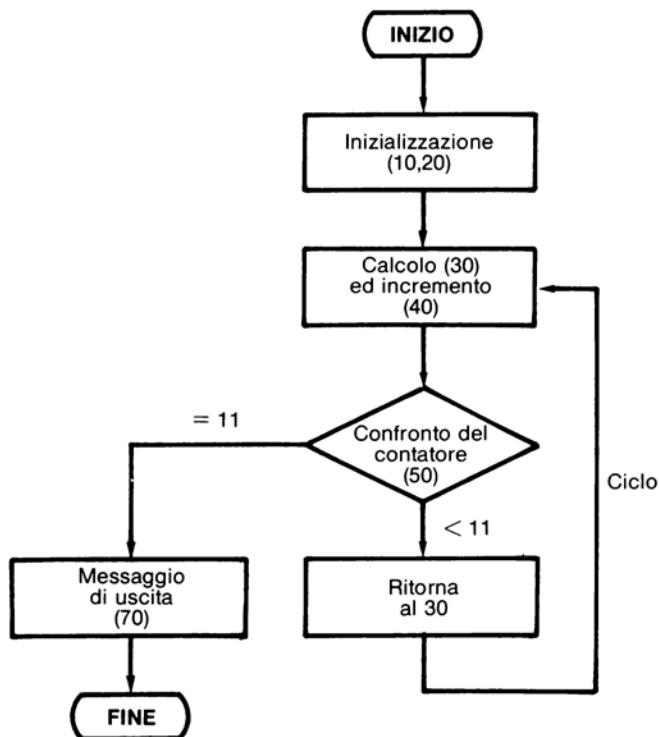



Figura 7.1 - Diagramma di flusso del programma SOMMA dei numeri interi.

Variazioni

Giocherelliamo ora con il nostro programma di addizione di interi per raffinare la nostra preparazione di programmazione. Quanto segue, dimostrerà le molte alternative che possono essere usate nello scrivere un programma. Per esempio, alla linea 50, potremmo aver scritto:

```
50  IF I >10 THEN 70
```

ed il risultato sarebbe stato lo stesso (quando I raggiunge il valore 11 è più grande di 10). Oppure, potremmo aver scritto:

```
40  IF I = 10 THEN 70
50  I = I + 1
```

invece di:

```
40  I = I + 1
50  IF I = 11 THEN 70
```

Con questa variazione, I viene prima controllata e poi incrementata. Osserva che questa volta I è confrontata con il valore 10 (invece che con 11).
Potremmo anche scrivere:

```
50  IF I<11 THEN 30
60  REM
```

Puoi verificare che tutte queste versioni sono corrette e che tutte queste variazioni sono valide ed equivalenti.
Anche un programma tanto breve come il programma SOMMA può essere scritto in molti modi equivalenti. Non esiste una via unica nello scrivere programmi; come in una lingua parlata si può esprimere lo stesso concetto in molti modi diversi.
Questo breve programma ha illustrato l'uso dei cicli e del contatore. Abbiamo analizzato anche le fasi tipiche di tali programmi: inizializzazione, calcolo, incrementazione, controllo e uscita. In vista di un uso frequente dei cicli nei programmi, è stata prevista una speciale istruzione per facilitare il loro uso nel BASIC, l'istruzione FOR...NEXT.

L'istruzione FOR...NEXT

L'istruzione FOR...NEXT automatizza la maggior parte delle istruzioni richieste per un loop. Spiegheremo il loro uso e le operazioni usando gli esempi attuali. Ecco un modo di riscrivere il nostro programma di addizione usando le nuove istruzioni:

```
1  REM * * * ADDIZIONI DI NUMERI INTERI- VERSIONE 2 * * *
10 SOMMA = 0
20 FOR I = 1 TO 10
30 SOMMA = SOMMA + I
40 NEXT I
50 PRINT "LA SOMMA DEI PRIMI 10 NUMERI INTERI E':"; SOMMA
60 END
```

Osserva che questo programma ha due istruzioni in meno del precedente; è più corto e più leggibile. Esaminiamolo in dettaglio:
La prima istruzione eseguibile inizializza SOMMA a zero:

```
10  SOMMA = 0
```

L'istruzione seguente è l'istruzione FOR:

```
20  FOR I = 1 TO 10
```

Questa istruzione ha molti compiti:

- Segna l'inizio di un loop automatico (si trova dove inizia il ciclo)
- Specifica che I (il contatore) inizia con il valore iniziale 1 quando questa istruzione viene eseguita. Ciò elimina la necessità di una istruzione di inizializzazione per I.
- I viene incrementato di 1 (fino al massimo di 10) ogni volta che l'istruzione viene riattivata dopo aver incontrato una istruzione NEXT. Il confronto viene eseguito automaticamente e quando I oltrepassa il valore 10, il ciclo non viene più eseguito e vien eseguita l'istruzione seguente il NEXT (uscita dal loop).
  Il corpo del ciclo contiene semplicemente l'accumulazione della somma:

```
30  SOMMA = SOMMA + I
```

L'istruzione NEXT:

```
40  NEXT I
```

segna la fine del loop e causa la riattivazione del FOR.
Questa rimpiazza due istruzioni della versione precedente:

```
40  I = I + 1

50  GOTO 30
```

Ogni volta che viene eseguito il NEXT I, il programma salta all'inizio del loop (cioè all'istruzione FOR) e quando FOR è riattivato:

- I è incrementato di 1
- Il nuovo valore viene confrontato con 10.

e finchè I non supera il valore 10, l'esecuzione va avanti. Il ciclo ha termine quando I è uguale a 10 e viene incontrata l'istruzione NEXT. A questo punto, avviene l'uscita e viene eseguita l'istruzione 50 (cioè quella che segue il NEXT). Questa sequenza è illustrata in Figura 7.2 (un diagramma di flusso).
La figura 7.2 mostra che l'istruzione FOR automatizza tre compiti:

- inizializzazione del contatore (I è posto inizialmente uguale a 1).
- incremento del contatore (I è incrementato ogni volta di 1).
- confronto del contatore con il valore massimo (I è confrontato con 10).

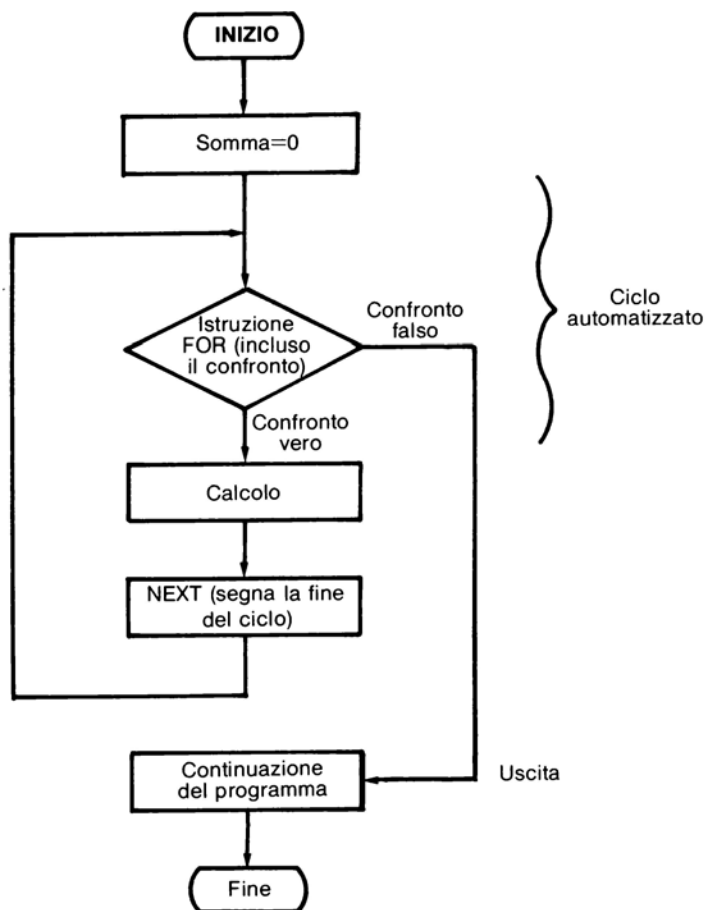



Figura 7.2 - Ciclo automatico con FOR...NEXT.

L'istruzione NEXT segna semplicemente la fine del ciclo ed esegue un "GOTO all'istruzione FOR". Dopo aver usato l'istruzione FOR alcune volte, certamente apprezzerai come semplifica la scrittura dei programmi e li rende più chiari.
FOR...NEXT è una istruzione molto utile; non hai l'obbligo di usarla, però probabilmente la troverai molto preziosa. Generalmente rende i programmi più chiari e minimizza il rischio di errore. Forniremo ora degli esempi pratici per illustrare l'uso dell'istruzione FOR...NEXT e l'uso dei loop automatici.

Somma dei primi N numeri interi

Calcoliamo ora la somma dei primi N numeri interi. Questa volta l'utente specifica il valore N tramite la tastiera.
Ecco il programma:

```
10  REM * SOMMA DEI PRIMI N NUMERI INTERI *
20  SOMMA = 0
30  INPUT "SOMMERO' I PRIMI N NUMERI INTERI. SCRIVI N:"; N
40  FOR I = 1 TO N
50  SOMMA =SOMMA + I
60  NEXT I
70  PRINT "LA SOMMA DEI PRIMI"; N;"NUMERI INTERI E' ";SOMMA
80  END
```

e l'esecuzione:

```
SOMMERO' I PRIMI N NUMERI INTERI. SCRIVI N:?5
LA SOMMA DEI PRIMI 5 NUMERI INTERI E' 15
```

Dovresti apprezzare la leggibilità del programma. Questa volta il ciclo viene eseguito da 1 ad N, dove N viene introdotto attraverso la tastiera (istruzione 30). Puoi migliorare questo programma effettuando il controllo dell'input: N dovrebbe essere maggiore di 1. L'interprete BASIC verificherà automaticamente che N sia un numero intero quando verrà eseguita l'istruzione FOR. Prova ad imbrogliarlo.

Tabella di valori

Usando la potente istruzione FOR...NEXT, mostreremo quanto è facile automatizzare i calcoli e visualizzare tabelle di valori. Ecco una tabella di elevazione al quadrato (un numero moltiplicato per se stesso) e al cubo (un numero moltiplicato per se stesso e poi di nuovo per se stesso:

```
10  REM TAVOLA DI QUADRATI E CUBI
20  REM DEI PRIMI 10 NUMERI INTERI
30  FOR I = 1 TO 10
40  PRINT I, I ^ 2, I ^ 3
50  NEXT I
60  END
```

ed ecco il risultato:

```
1      1      1
2      4      8
3      9      27
4      16     64
5      25     125
6      36     216
7      49     343
8      64     512
9      81     729
10     100    1000
```

Esaminiamo più dettagliatamente l'istruzione 40:

40 PRINT I,I^2, I^3

I^2 significa I elevato alla 2 potenza, cioè I*I. Per esempio, se I=2, allora I^2=2x2=4. Ugualmente, I^3 significa I elevato alla 3 potenza, cioè I*I*I. Se I=4, allora I^3=4x4x4=64.
Osserva che questa volta usiamo una virgola nell'istruzione PRINT cosicchè i risultati sono accuratamente allineati quando vengono visualizzati. La virgola porta ad eseguire una spaziatura automatica (o *tabulazione*) sulla linea. L'esatta spaziatura tra le colonne dipende dal tuo interprete BASIC. Per esempio, può essere di 14 caratteri.
Come esercizio, puoi riscrivere questo programma per visualizzare la somma dei quadrati e dei cubi dei primi N numeri interi, dove N è letto dalla tastiera. Impareremo come farlo in un paragrafo seguente.

Linee di asterischi

Ecco un semplice programma che visualizza N linee di asterischi, dove N è un numero che specifichi tramite la tastiera:

```
10   REM * LINEE DI ASTERISCHI *
20   PRINT "VISUALIZZERO' LINEE DI ASTERISCHI"
30   INPUT "DIMMI QUANTE LINEE :"; N
40   REM N E' IL NUMERO DELLE LINEE DA VISUALIZZARE
50   FOR I = 1 TO N
60   PRINT "****************************************"
70   NEXT I
80   END
```

ed ecco l'esecuzione:

Di nuovo, ogni volta che un input viene introdotto attraverso la tastiera, è buona norma controllarlo per evitare strani comportamenti del programma. Ci aspettiamo che colui che lo usa introduca un numero positivo ed assumiamo che tu non voglia visualizzare più di 20 linee.
Dovrai avvertire di ciò l'utente tramite una istruzione PRINT e controllare l'input con la seguente istruzione:

```
IF (N<1) OR (N>20) THEN GOTO 20
```

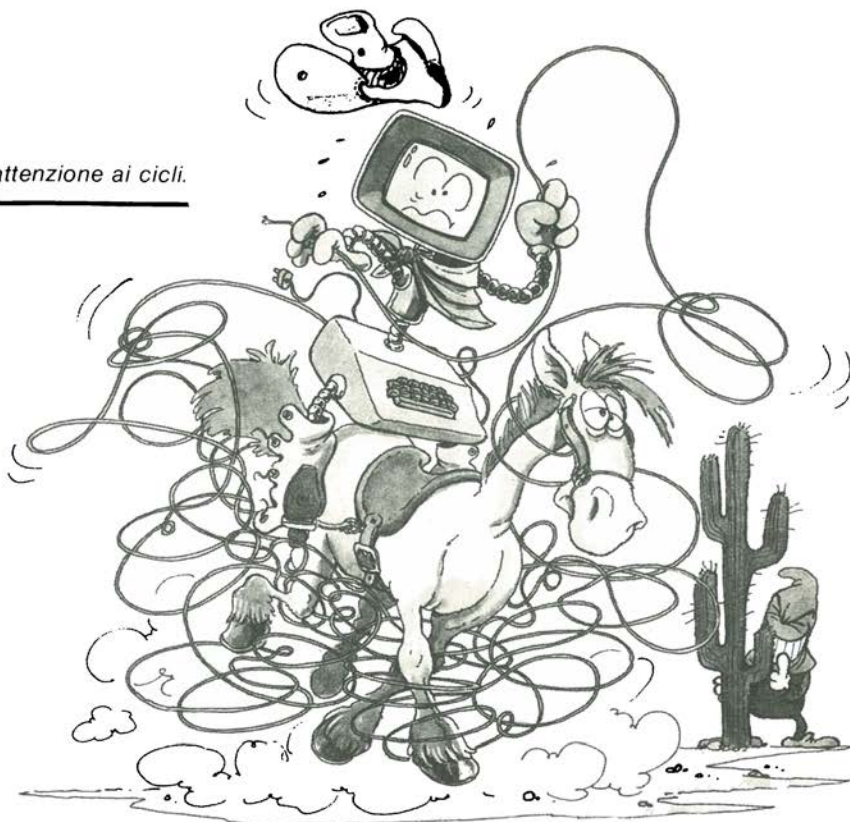
Cicli più avanzati

L'istruzione FOR...NEXT offre altri due vantaggi che non abbiamo ancora descritto:

- Puoi incrementare il contatore di qualsiasi valore intero come 2, 3, 4 o anche —1 (invece di 1). Questo è chiamato *passo variabile*.

- Puoi creare un loop dentro l'altro. Tali cicli sono chiamati *cicli nidificati*.

Esaminiamo queste due tecniche.

Fai attenzione ai cicli.

Passo variabile

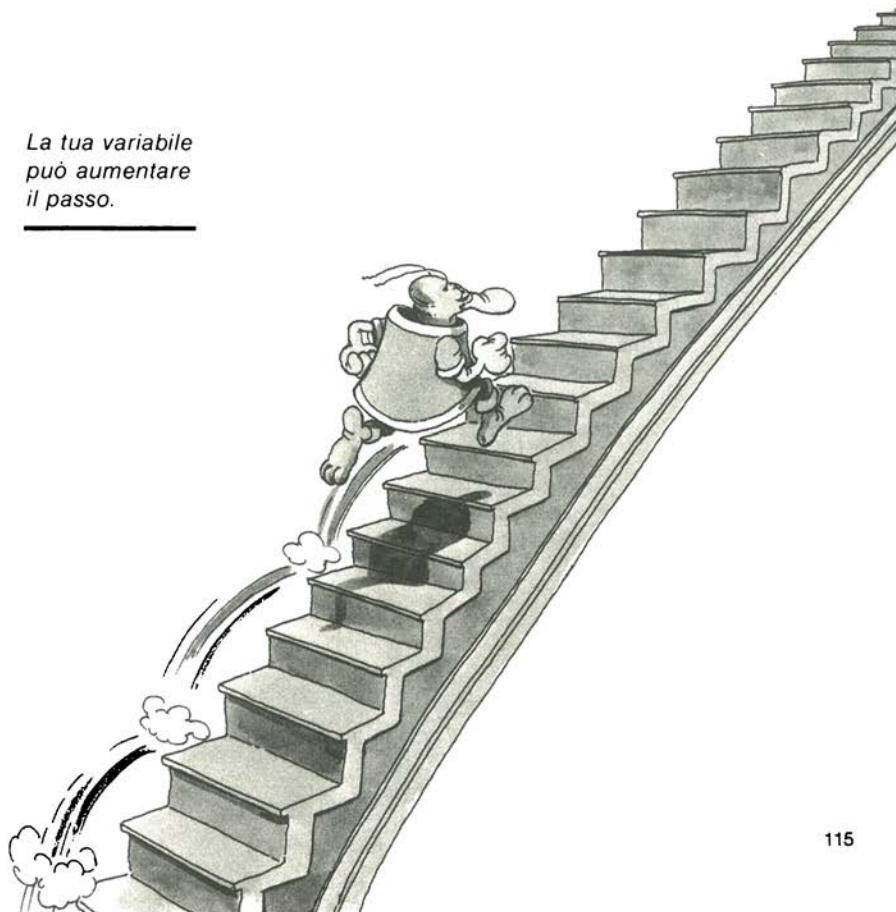
Ecco un esempio di passo variabile:

```
FOR I=1 TO 5 STEP 2
```

Ogni volta che il loop viene iniziato di nuovo, I verrà incrementato di 2. Potresti scrivere anche:

```
FOR I=10 TO —5 STEP —1
```

usando un *passo negativo*. Poichè il limite superiore del contatore (qui —5) è minore del valore iniziale (10), per l'interprete è un "passo negativo". Il valore di I viene decrementato di 1 ogni volta. Il primo valore di I sarà 10, il successivo 9, il successivo 8 e l'ultimo sarà —5. In altre parole verrà generata la seguente sequenza di numeri: 10, 9,8, 7, 6, 5, 4, 3, 2, 1, 0, —1, —2, —3, —4, —5. Il passo negativo è un'altra caratteristica conveniente che potrai desiderare di utilizzare.

La tua variabile può aumentare il passo.

La tecnica dei cicli nidificati è una caratteristica importante e potente, usata per automatizzare elaborazioni complesse. Un ciclo nidificato è creato quando usi un gruppo di istruzioni FOR...NEXT dentro un loop; cioè dentro un altro gruppo di istruzioni FOR...NEXT.
Come regola generale, tra una istruzione FOR e un NEXT puoi usare quante istruzioni desideri. In modo specifico, puoi includere tra queste istruzioni anche un altro ciclo. Quando accade ciò si ha un ciclo nidificato. Questo concetto è illustrato in figura 7.3. Tieni presente che quando usi dei loop nidificati, il programma diviene più difficilmente leggibile. Per rimediare a ciò sei invitato a curare l'impaginazione del programma. Una impaginazione curata è un'altra tecnica per rendere più chiari i programmi. La figura 7.4 mostra una versione con impaginazione curata della Figura 7.3.

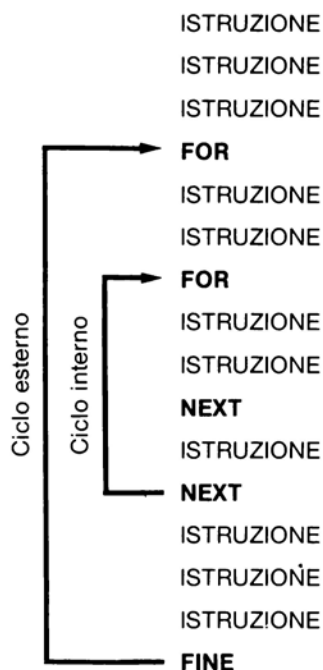
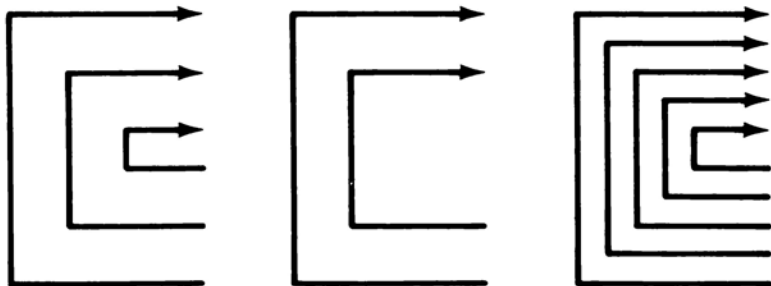
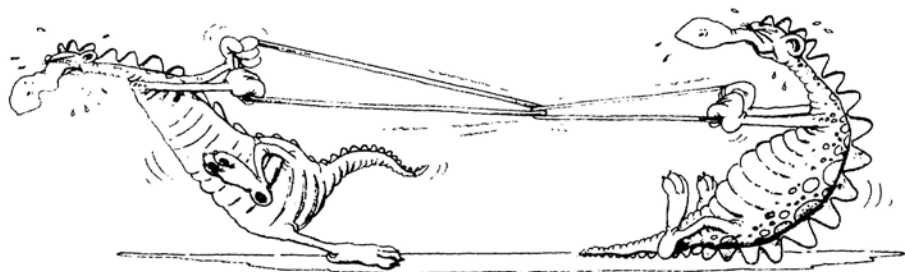
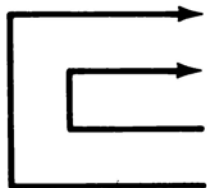
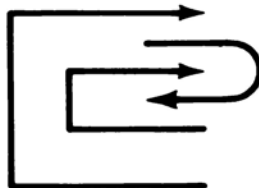
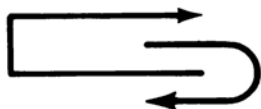
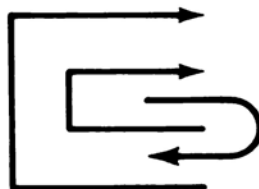



Figura 7.3 - Un ciclo nidificato.

Figura 7.4 - Una impaginazione migliorata.

Puoi nidificare i loop a qualsiasi livello fino ad un massimo imposto dal tuo interprete o dalla dimensione della memoria. Comunque non puoi intrecciare i cicli. I seguenti cicli sono permessi:

ed i seguenti sono errati:

Non intrecciare i cicli!

Inoltre non puoi saltare (con un GOTO) da un punto dentro un loop più esterno ad un punto dentro un loop più interno:

Nidificazione corretta | **Salto non permesso (tramite IF o GOTO)**

Invece puoi farlo da uno più interno:

Salto corretto | **Salto corretto**

Ecco un esempio di ciclo nidificato. Questo programma visualizza il tempo (accellerato) in minuti ed ore:

```
10 REM * * * OROLOGIO SIMULATO * * *
20 FOR ORA = 0 TO 23
30 FOR MINUTO = 0 TO 59
40 PRINT "L'ORARIO E' "; ORA; "ORE E "; MINUTO; "MINUTI"
50 NEXT MINUTO
60 NEXT ORA
70 PRINT "FINE DELLA GIORNATA"
80 END
```

Ecco lo schermo durante l'esecuzione:

```
L'ORARIO E' 0 ORE E 0 MINUTI
L'ORARIO E' 0 ORE E 1 MINUTI
L'ORARIO E' 0 ORE E 2 MINUTI
L'ORARIO E' 0 ORE E 3 MINUTI
L'ORARIO E' 0 ORE E 4 MINUTI
                    .
                    .
                    .
L'ORARIO E' 0 ORE E 59 MINUTI
L'ORARIO E' 1 ORE E 0 MINUTI
```

Caratteristiche addizionali

Come nota finale, i numeri decimali e le espressioni sono generalmente permessi nelle istruzioni FOR. Per esempio, sono valide le seguenti istruzioni:

FOR MISURA = 0.1 TO 13.5 STEP 0.2

FOR NUMERO = N TO (N*2) STEP 1

Questa tecnica non è consigliata ed in questo libro non la useremo.

Riassunto

I cicli sono usati estesamente per automatizzare la ripetizione di una parte del programma. L'istruzione FOR...NEXT è usata in BASIC per automatizzare i loop. Nella maggioranza dei casi l'istruzione FOR...NEXT può sostituire parecchie altre istruzioni BASIC. In questo capitolo, abbiamo esaminato le tipiche utilizzazioni dell'istruzione FOR...NEXT, inclusi i cicli nidificati ed abbiamo sviluppato parecchi programmi avanzati.
Ora hai imparato tutte le tecniche di base della programmazione e sei pronto per iniziare a scrivere programmi tuoi. Nel prossimo capitolo spiegheremo come iniziare.

Esercizi

7.1: Visualizza i primi 15 numeri interi sulla stessa linea (4 istruzioni).

7.2: Scrivi un programma che legga dalla tastiera il tempo in ore e minuti e lo visualizzi come segue:

Input:	3 *(ore)*	5 *(minuti)*
video:	HHH	3 *(lettere)*
	MMMMM	5 *(lettere)*

(Suggerimento: Puoi eseguire PRINT LETTERA$; ripetutamente per visualizzare diversi caratteri).

7.3: Che cosa è un contatore in un loop?

7.4: Puoi saltare nel bel mezzo di un ciclo?

7.5: Visualizza una tabella che converta once in grammi (1 oncia = 28 grammi)

7.6: Calcola la somma dei primi N numeri interi dispari (dove N è immesso da tastiera) e visualizzala per ogni numero intero.

7.7: Leggi i punteggi di cinque studenti che hanno sostenuto quattro prove (voti da 0 a 10). Visualizza i voti, il totale e la media per ogni studente.

7.8 Visualizza una tabella delle percentuali IVA per prezzi da 1 a 100 mila lire con incrementi di mille lire. La percentuale IVA deve essere introdotta da tastiera.

Capitolo **8**

Creare un programma

Programmare significa progettare un programma che automatizzi un certo lavoro. Finora abbiamo scritto molti brevi programmi senza passi intermedi, direttamente scrivendo una sequenza di istruzioni BASIC. Questa tecnica può andar bene per i programmi molto semplici, ma non va per quelli complessi.
In questo capitolo, impareremo la maniera corretta di creare un programma.

Essa si compone di cinque passi:

1. **Indica esattamente la sequenza dei passi che la soluzione del problema richiede. Questa fase è la progettazione dell'***algoritmo*.
2. **Disegna un diagramma che mostri la sequenza degli eventi e dei passi logici. Questa fase è il progetto del** *diagramma di flusso* **(flowchart).**
3. **Scrivi il programma in BASIC. Questa fase è chiamata** *codificazione*.
4. **Verifica e prova il programma. Questa fase è chiamata** *debugging*.
5. **Rendi il programma chiaro e documentato. Questa fase è chiamata** *documentazione*.

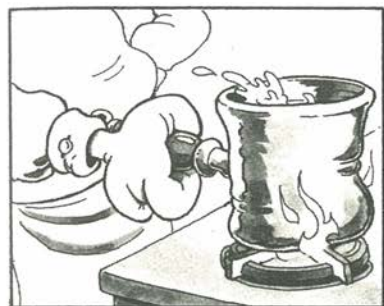
Finora abbiamo imparato e messo in pratica solamente i passi da 2 a 5, ma questa sequenza va bene soltanto per programmi brevi. Prima di passare ai programmi lunghi, studiamo la sequenza completa richiesta per lo sviluppo dei programmi.

Progetto dell'algoritmo

Desideriamo progettare un programma che risolva un dato problema o automatizzi un certo lavoro. Finora abbiamo studiato programmi che risolvono semplici problemi. La sequenza dei passi richiesta per risolvere ogni problema era talmente banale che non c'era quasi nessuna fase da studiare. Come caso generale, comunque, dato un problema dobbiamo prima studiarne la soluzione. Per poter scrivere un programma, la soluzione deve essere descritta come una sequenza di passi. La sequenza dei passi è detta *algoritmo*. Formalmente, un algoritmo è definito come una descrizione passo-passo della soluzione di un problema. Tecnicamente, un algoritmo deve anche arrestarsi, non dovrebbe continuare indefinitamente. Un algoritmo che non si arresta è un *errore*!
Ecco un problema semplice: convertiamo un peso misurato in once nel suo equivalente in grammi. Ricorda che una oncia equivale a 28.35 grammi. La soluzione è banale: dobbiamo moltiplicare il peso in once per 28.35. Questo è un semplice algoritmo ad un solo passo.
Esaminiamo ora un problema leggermente più complesso: leggiamo un numero dalla tastiera e verifichiamo che sia in un certo intervallo. Accetteremo come numeri validi, quelli che appartengono all'intervallo tra 0 e 100. La sequenza dei passi richiesta per la soluzione di questo problema è la seguente:

1. Leggi un numero.
2. Verifica che sia maggiore di zero. Se è così vai avanti, altrimenti rifiuta il numero.
3. Verifica che sia minore di 100. Se sì, accetta il numero, altrimenti rifiutalo.

Questo era un algoritmo composto di tre passi.
In pratica, la maggior parte dei problemi sono molto più complicati e richiedono algoritmi più lunghi e molto più complessi. Ecco alcuni esempi quotidiani di algoritmi. Ne potrai trovare molti di più nel tuo libro di cucina o nella tua auto o nei manuali degli elettrodomestici.
Esaminiamo un algoritmo per cuocere un "uovo in tre minuti". Ecco i passi:

1. Prendere una pentola.
2. Riempirla d'acqua.
3. Accendere il fornello.
4. Mettere la pentola piena d'acqua sul fornello.
5. Portare l'acqua all'ebollizione.
6. Mettere un uovo nell'acqua bollente.
7. Far partire il timer per tre minuti.
8. Quando il timer suona, togliere l'uovo.
9. Spegnere il fornello.

"Dimostrerò l'algoritmo dell'uovo cotto in tre minuti".

Questo algoritmo sembra semplice, ma se dovesse essere eseguito da un robot computerizzato, dovrebbe essere molto più preciso. Per esempio, dovresti specificare esattamente quale pentola usare e con precisione quanta acqua mettere nella pentola. Molti algoritmi presentati nei libri di tutti i giorni, presumono che l'utente abbia una certa esperienza tecnica e culturale e perciò sono generalmente incompleti. In altre parole, presumono che l'utente possa coprire le lacune.
Sfortunatamente, questa è proprio la ragione per cui certi manuali sono così difficilmente comprensibili!
Non faremo qui lo stesso errore. I nostri algoritmi saranno completamente precisi per divenire poi programmi eseguibili.
Ecco un ultimo esempio: un algoritmo per far partire l'auto. Se presumiamo che l'auto funzioni perfettamente, l'algoritmo è molto semplice:

1. Inserisci la chiave d'accensione.

2. Gira la chiave completamente a destra.

3. Rilascia la pressione sulla chiave mentre applichi una leggera pressione sul pedale dell'accelleratore.

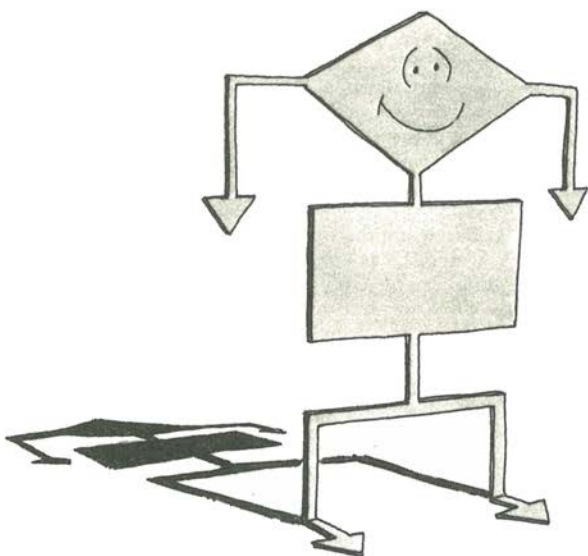
Far partire un'auto è un algoritmo interessante.

Sappiamo, comunque, che l'auto potrebbe anche non partire. Questo accade perchè concorrono molti altri fattori, come la temperatura o le condizioni del motore. Preparare un algoritmo per far partire un'auto che tenga conto di tutte le condizioni, richiederebbe numerose pagine, se tenessimo conto di tutte le cose che potrebbero non funzionare.

Nella vita quotidiana, possiamo semplificare i passi di un algoritmo, ma, in un computer, questo non è possibile.
Un algoritmo deve essere preciso e completo.
Quando progetti un algoritmo per una soluzione tramite il computer, devi essere meticoloso ed anticipare ogni situazione dovesse presentarsi; altrimenti il tuo programma potrà sbagliare. I programmi che funzionano richiedono una specifica attitudine: devi continuamente mettere in dubbio ogni cosa che fai, presumendo sempre che ci possa essere qualcosa di sbagliato o di incompleto.

Ricorda che si suppone che l'algoritmo funzioni!'

Non presumere mai che un input possa essere logico.
Provalo e verificalo. È sempre possibile commettere un errore. Illustreremo queste considerazioni in seguito nel prossimo capitolo quando esamineremo un argomento di studio attuale.

Insomma, per automatizzare la soluzione di un problema scrivendo un programma per computer, comincia con il preparare un algoritmo.L'algoritmo finale dovrebbe essere perfetto anche se all'inizio difficilmente lo è. Infatti, probabilmente scriverai una soluzione approssimata (un algoritmo impreciso) e poi continuerai con il rifinire l'algoritmo finchè non raggiunga uno stato che considererai perfetto. Assicurati sempre che il tuo algoritmo sia perfetto prima di iniziare a scrivere le istruzioni.

"Io sono Flowchart,
il diagramma di flusso.
Voglio aiutarti,
ma per piacere, utilizzami".

Costruzione di un diagrama di flusso (flowchart)

Ora che abbiamo progettato l'algoritmo, il pensiero che viene alla mente è: traduciamolo subito in un programma BASIC ed eseguiamolo! Sbagliato.
C'è ancora un passo che ti risparmierà molte ore di programmazione: la costruzione del diagramma di flusso. Se salterai questa fase, probabilmente non scriverai un programma che funziona e perderai molto tempo più tardi cercando di correggere il programma, senza alcuna garanzia di successo.
Per contrasto, una volta che hai un buon diagramma di flusso, scrivere il programma è un passo semplicissimo.

Un diagramma di flusso è semplicemente un diagramma che mostra la sequenza degli eventi. La Figura 8.1 mostra un diagramma di flusso dei passi necessari per cuocere "un uovo in tre minuti".

Come puoi vedere, questo diagramma di flusso è una rappresentazione grafica dell'algoritmo che abbiamo già presentato. In ogni caso, ogni rettangolo rappresenta un passo dell'algoritmo. Lo scopo del diagramma di flusso è di mostrare ulteriormente la sequenza dei passi. Più tardi, quando avrai preso pratica con i diagrammi di flusso,

Figura 8.1 Cottura di un uovo in tre minuti.

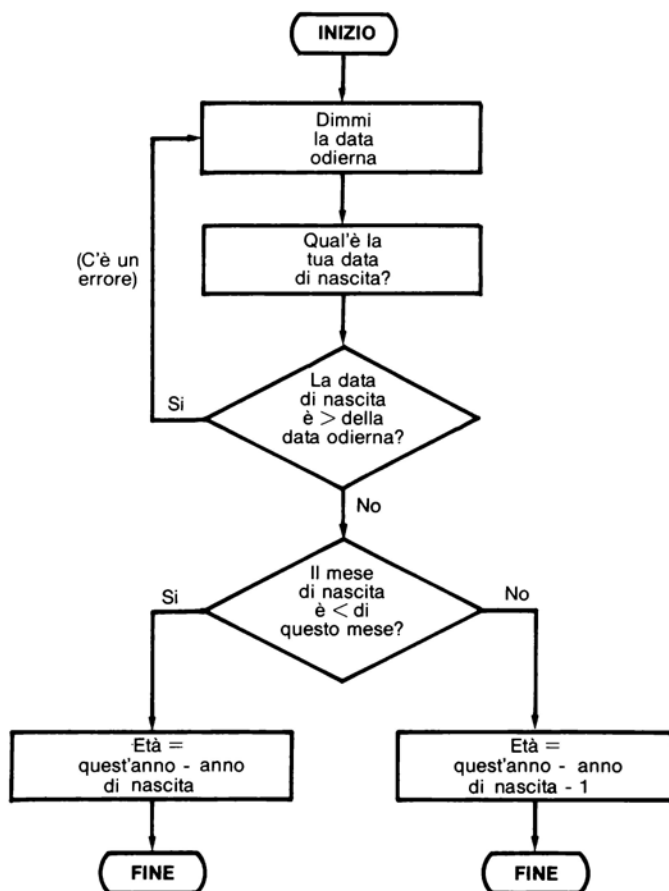



Figura 8.2 - Diagramma di flusso per il calcolo dell'età.

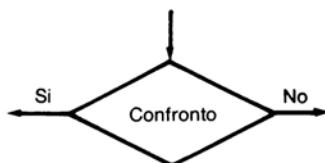
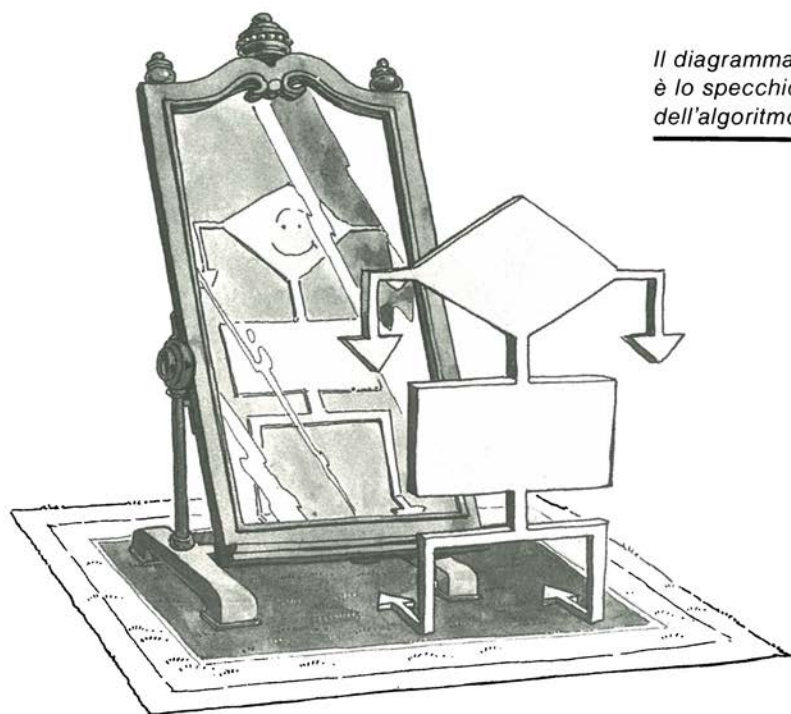



Figura 8.3 - Simbolo a forma rombica

potrai anche saltare la progettazione dell'algoritmo ed iniziare direttamente con il diagramma di flusso, poichè il diagramma di flusso è la giusta rappresentazione dell'algoritmo.
Nel caso di un algoritmo semplice come quello per la cottura di un uovo, il diagramma di flusso non è molto utile e se ne potrebbe fare anche a meno. Il vero valore di un diagramma di flusso si manifesta quando inizierai a scrivere algoritmi più complessi che coinvolgono scelte e decisioni.

Scriviamo ora un nuovo programma che chieda la tua data di nascita e la data odierna per calcolare la tua età.
L'algoritmo è banale ed il diagramma di flusso è mostrato in Figura 8.2. Gli elementi di forma rombica nel diagramma di flusso indicano un confronto, cioè una scelta nella sequenza. A questo punto, per facilitare la conversione del diagramma di flusso in un programma, assicurati che ogni scelta abbia *due* risultati: "si" o "no".
Contrassegna conseguentemente le frecce. Esaminiamo ora la Figura 8.2 e verifichiamo che ogni freccia che esce da un elemento rombico sia contrassegnata con "si" o "no" a seconda del risultato del confronto (vedi Figura 8.3).

Il diagramma di flusso è lo specchio dell'algoritmo.

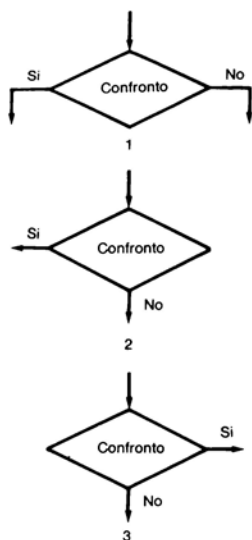



Figura 8.4 - I tre modi di disegnare le frecce.

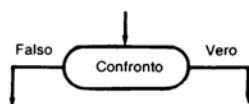



Figura 8.5 - Un'altra forma di un elemento decisionale.

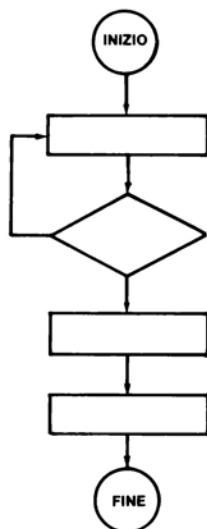



Figura 8.6 I simboli di inizio e fine.

Generalmente ci sono tre modi di disegnare le frecce, come indicato in Figura 8.4 e puoi scegliere quello che preferisci. La tua scelta dovrebbe essere fatta in modo da facilitare la lettura del diagramma di flusso. La posizione dei "si" e dei "no" può essere cambiata liberamente; in altre parole, se preferisci, "si" può stare sulla destra.

Torniamo alla Figura 8.2 ed al calcolo del diagramma di flusso e all'algoritmo che rappresenta.
Inizialmente è richiesta la data odierna. Questa fase corrisponde al primo elemento (contrassegnato 1). In secondo luogo è richiesta la tua data di nascita. (elemento contrassegnato con 2 sul diagramma di flusso). Poi, dobbiamo verificare che la data di nascita introdotta sia ragionevolmente possibile. Dobbiamo verificare che la data di nascita sia precedente alla data odierna. Se il valore della tua data di nascita è maggiore del valore della data odierna, viene riconosciuto come errore e si ricomincia daccapo; altrimenti la data di nascita viene assunta essere valida. Questo passo corrisponde all'elemento rombico (contrassegnato con 3). Per essere ancora più precisi, potremmo anche non accettare date di nascita che comportino una età maggiore di 150 anni, poichè estremamente improbabili. Comunque, accettare queste date non porta a seri problemi perciò non vale la pena di verificarle.

Nell'elemento 4 del diagramma di flusso, verifichiamo che il mese della tua nascita sia minore del mese odierno. Se è così, il tuo compleanno è già passato e la tua età può essere calcolata (elemento 5) come differenza tra l'anno corrente e l'anno di nascita. Per esempio, se sei nato nel Febbraio 1946 ed ora siamo in Marzo 1983, la tua età è 1983-1946=37.

Altrimenti (elemento 6 del diagramma), la tua età è calcolata come la differenza tra l'anno corrente e l'anno di nascita meno 1. Per esempio, se tu fossi nato nel Giugno 1942 ed ora siamo in Marzo 1983, la tua età sarebbe 1983-1942-1=40. Per mantenere semplice questo programma, non confrontiamo i giorni del mese. Aggiungeremo questo miglioramento in seguito.
I passi di un algoritmo dovrebbero ora essere chiari.
Passiamo quindi ad esaminare i simboli del diagramma di flusso.

I simboli nei diagrammi di flusso

In un diagramma di flusso, gli elementi di forma rettangolare sono usati per il calcolo e per le azioni dirette che non comportano una scelta; come un input o una visualizzazione. Gli elementi di forma rombica sono usati per i confronti o per le scelte e da essi devono uscire due frecce. Inoltre tutti gli algoritmi devono avere un inizio ed una fine che vengono indicati con INIZIO e FINE.
I simboli usati nei diagrammi di flusso non sono standardizzati. Sono stati proposti molti standard, ma nessuno è mai stato accettato universalmente. Gli elementi a forma rettangolare però sono usati sempre. Quello a forma rombica, invece, può essere sostituito con uno con gli spigoli arrotondati come mostrato in Figura 8.6. Per finire, possono venir usati dei simboli speciali per indicare l'uso di periferiche specifiche. Per esempio una operazione PRINT può essere rappresentata in uno dei due modi indicati in Figura 8.7.
In pratica non devi preoccuparti dei simboli. Un diagramma di flusso è un modo semplice per una utile visualizzazione di un algoritmo (specialmente quando contiene molte scelte). Potresti anche usare, volendo, altri simboli, ma è preferibile usare questi più comuni, in modo che i tuoi programmi possano essere uniti più facilmente con altri e tu possa leggere e seguire più facilmente altri diagrammi di flusso.

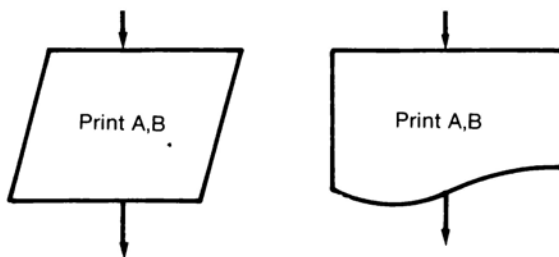



Figura 8.7 - I simboli per la visualizzazione o stampa (PRINT).

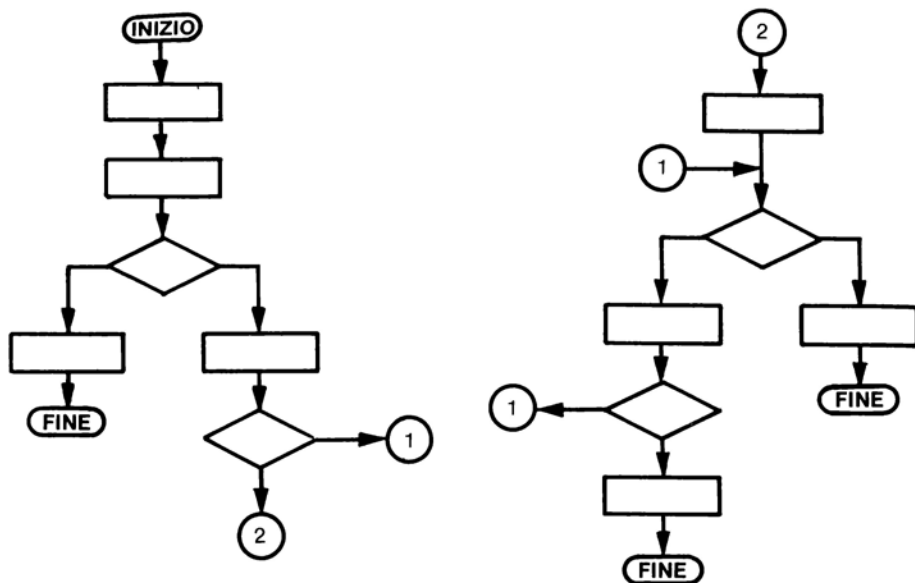



Figura 8.8 - Suddivisione di un diagramma di flusso.

Dividere il diagramma di flusso

Ecco una convezione più utile. Se un diagramma di flusso si estende per più pagine, si può dividerlo in parti.
Contrassegna ogni punto di divisione con un nome o un numero ed assicurati di avere il corrispondente ingresso del diagramma in un'altra pagina. La Figura 8.8 mostra un esempio dell'uso dei numeri per la connessione dei punti.

Rifinire il diagramma di flusso

Le istruzioni poste dentro gli elementi di un diagramma di flusso, possono essere scritte a tuo piacimento, perchè esse non sono delle istruzioni BASIC. Quando scrivi un diagramma di flusso la prima volta, potrai scrivere istruzioni molto vaghe come "dimmi la tua data di nascita?".
In seguito potrai rifinire le istruzioni contenute negli elementi e scrivere più dettagliatamente il diagramma che poi verrà più facilmente tradotto in programma.
Se credi che le istruzioni poste dentro gli elementi siano sufficientemente precise per scrivere un programma equivalente, non è necessario che le modifichi ancora. Se invece, ritieni che esse siano troppo vaghe o complesse per essere tradotte direttamente in un programma, dovresti allora sostituirle con una sequenza di istruzioni più dettagliate.

Come esempio, puoi ricordare che il diagramma di flusso di Figura 8.2 confrontava il mese di nascita ma non il giorno, per determinare se il tuo compleanno questo mese sia già passato. Rifiniamolo per confrontare oltre al mese anche il giorno.Il corrispondente segmento del programma iniziale (non rifinito) è in Figura 8.9, mentre quello nuovo è rifinito appare in figura 8.10.

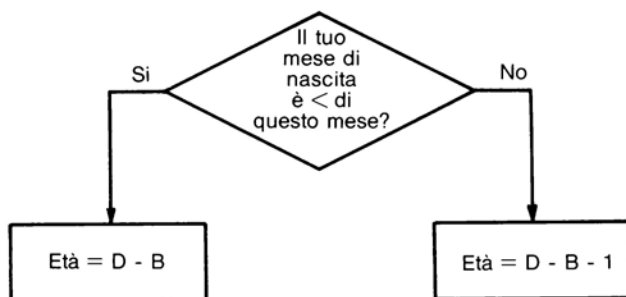



Figura 8.9 - Un algoritmo approssimato .

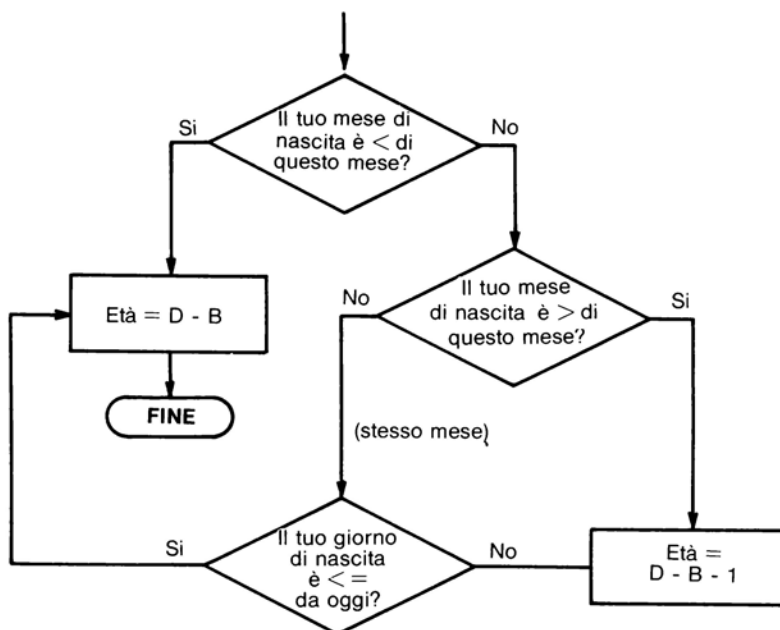
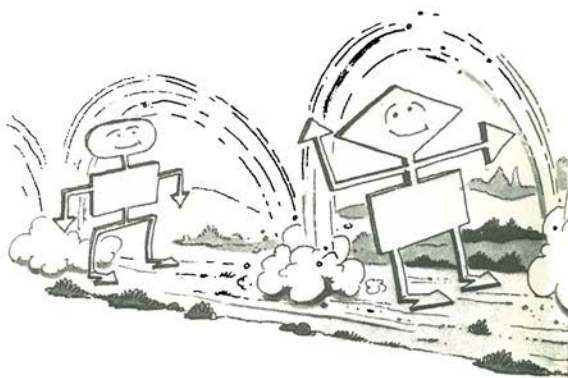



Figura 8.10 - Un diagramma di flusso rifinito.

In pratica, la maggior parte delle persone non scrivono l'algoritmo, ma passano direttamente alla fase di costruzione del diagramma di flusso. Questo va bene. Invece, qualche volta i programmatori esperti (ed anche non tanto esperti) saltano anche questa fase e passano direttamente alla scrittura del programma. Questo è altamente pericoloso ed incline agli errori. Ti raccomando fortemente di disegnare il diagramma di flusso prima di scrivere ogni programma. In seguito, quando sarai un programmatore esperto, sarai capace di evitare i dettagli sul diagramma di flusso e disegnarne solamente uno approssimativo.

Collaudo manuale

Una volta che hai scritto un diagramma di flusso, provalo con esempi pratici ed assicurati che i risultati siano corretti o almeno sembrino corretti. Questa fase viene chiamata collaudo *manuale*, perchè senza l'uso del computer. Per esempio, ritorna al diagramma di flusso del calcolo dell'età ed introduci la data odierna e la tua data di nascita, come indicato. Il risultato è la tua età esatta? Se sì, le cose vanno bene.

Se salterai il diagramma di flusso, probabilmente cadrai in errore.

Se no, c'è un errore. Ora prova nuovamente con valori diversi, usando date di nascita che cadano prima o dopo della data odierna. Funziona ancora? Se sì, l'algoritmo è probabilmente corretto. Se no, c'è un errore. Il collaudo manuale è un modo rapido per verificare che non esistano dei banali errori.
Con la scrittura di un diagramma di flusso che funzioni, abbiamo compiuto tutti i passi preliminari richiesti prima della scrittura dell'attuale programma. Scriviamo ora il programma.

Codificazione

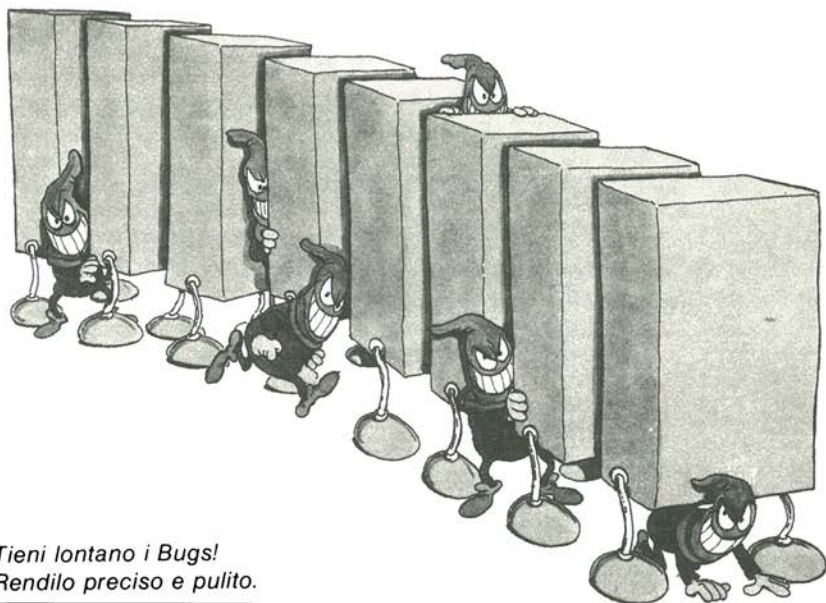
La scrittura delle istruzioni di un programma è detta *codificazione*; per *programmazione*, invece, normalmente si intende tutta la sequenza richiesta per creare un programma: studio dell'algoritmo, disegno del diagramma di flusso, codificazione, ricerca degli errori e collaudo finale. La codificazione comporta la traduzione del contenuto degli elementi del diagramma di flusso in istruzioni di programma espresse in un linguaggio programmativo, come il BASIC.
Questo è ciò che abbiamo imparato finora: abbiamo imparato a tradurre le istruzioni, le formule, i confronti e le condizioni in istruzioni BASIC.
Il segreto per una facile codificazione è di scrivere un diagramma di flusso sufficientemente dettagliato in modo da poter tradurre ogni singolo elemento del diagramma in poche istruzioni BASIC. Generalmente nei primi stadi di programmazione ogni elemento del diagramma di flusso viene tradotto in pochissime istruzioni, diciamo una o due, cioè c'è una corrispondenza diretta tra elementi ed istruzioni.
In seguito, quando crescono pratica ed esperienza, sarai in grado di scrivere diagrammi di flusso più "concentrati" ed ogni elemento verrà tradotto in molte istruzioni BASIC.
Contrariamente a quanto sembra, la codificazione è spesso una delle fasi che richiede minor tempo nella sequenza di sviluppo di un programma. Le prove di un programma richiedono, normalmente, molto più tempo della codificazione. Ecco perchè è importante scrivere un buon diagramma di flusso che minimizzi gli errori ed il tempo di prova.
Quando codifichi un programma, ricordati di farlo preciso, chiaro e leggibile, in modo che funzioni subito e possa essere facilmente provato e modificato.

"La codificazione è semplice quando si ha un buon diagramma di flusso!"

Scrivi il programma accuratamente

Scrivi il tuo programma con il massimo di attenzione, poichè ogni errore nella sistemazione di un segno di punteggiatura potrà far sì che il programma sbagli.

Scrivi il programma in modo chiaro

Usa i nomi delle variabili che siano facili da ricordare.
Lascia intervalli o anche salti nella sequenza dei numeri di linea nel caso che tu debba in futuro aggiungerci in mezzo delle istruzioni. Usa i commenti (REM) senza parsimonia in tutto il programma per rendere chiaro cosa sta facendo.
A questo punto, hai studiato l'algoritmo, hai disegnato il diagramma di flusso, hai scritto il programma e ti aspetti seriamente che funzioni. Invece no, mi spiace, ma nella maggior parte dei casi, *la prima volta il programma non funziona*. Generalmente ci vogliono parecchi tentativi ed un po' di esperienza prima che un programma di qualsiasi lunghezza funzioni correttamente.
Questo è l'argomento della prossima fase.

Tieni lontano i Bugs!
Rendilo preciso e pulito.

Eliminazione degli errori (debugging)

Hai codificato il tuo diagramma di flusso in un programma BASIC e, quindi, il tuo programma è sulla carta. Ora dovresti scriverlo sulla memoria del computer, scrivere "RUN" ed assicurarti che funzioni. Questa fase è detta *collaudo ed eliminazione degli errori o debugging.* Gli errori di un programma sono detti *bug.* L'eliminazione degli errori è detto *debugging.*
Ogni volta che trovi un errore devi correggerlo e dare nuovamente il RUN. Anche se sei stato particolarmente attento e meticoloso nello scrivere il tuo programma, un programma lungo difficilmente funziona la prima volta.
Questo accade perchè è molto facile durante la stesura del programma, mettere un carattere o anche un'intera frase al posto sbagliato. Anche i migliori programmatori devono perdere un certo tempo per correggere i loro programmi.
Perciò non ti sorprendere se dovrai correggere il tuo programma molte volte prima che funzioni.

L'interprete ti aiuta diagnosticando gli errori di sintassi.

Fortunatamente, il tuo interprete BASIC ti aiuterà nella diagnosi di alcuni problemi. Se il programma contiene un errore di scrittura che può essere diagnosticato dall'interprete, il programma si arresterà dopo che avrai dato il comando RUN e l'interprete ti darà un messaggio diagnostico del tipo "SYNTAX ERROR in line 84" (errore di sintassi alla linea 84). L'interprete ti sarà di molto aiuto nella ricerca degli *errori sintattici* (l'uso non permesso di simboli o operazioni), mentre, sfortunatamente, non ti aiuterà a trovare gli errori più pericolosi, gli *errori logici o di progetto.*
Questo è compito tuo. Ecco perchè devi investire il tuo tempo per scrivere attentamente il diagramma di flusso e perchè ogni volta che un numero viene introdotto alla tastiera o generato da essa, dovresti sempre controllare la sua validità confrontando il suo intervallo. Nel caso che il tuo programma abbia un difetto logico, questa tecnica ti aiuterà ad isolare la parte del programma dove è localizzato l'errore.

Generalmente, nel caso di semplici programmi, l'interprete BASIC rivelerà la presenza di alcuni errori tipografici che una volta corretti, permetteranno al tuo programma di funzionare. Dovrai assicurarti che il programma funzioni correttamente, facendo la prova in casi diversi o con valori diversi, perchè il tuo programma potrebbe contenere dei difetti logici. Nella maggior parte dei casi, però, troverai che il tuo programma funziona correttamente.

Alcuni consigli pratici

Ecco un consiglio pratico: dovresti inserire alcune istruzioni PRINT addizionali qua e là nel programma per verificare alcuni valori chiave. Ciò ti aiuterà a rivelare la presenza di valori strani e ad isolare le istruzioni che li causano. Ecco un esempio di PRINT che puoi inserire:

```
1235 PRINT "TEST PER IL VALOR MEDIO: "; MEDIA
```

Poi una volta che il programma funziona, puoi rimuovere questi PRINT in più. Questa tecnica è il *tracing o tracciamento* di una variabile.
Un altro consiglio pratico: ogni volta che il programma si arresta per conto suo o a causa dell'interprete, dovresti usare *l'esecuzione in modo immediato* per verificare il valore di alcune variabili nel tuo programma.
Per esempio, puoi scrivere:

```
>PRINT MEDIA
```

poi:

```
>PRINT SOMMA
```

per sondare il valore corrente di queste due variabili.
Il segreto di un debugging ben eseguito è l'esperienza ed un po' di prevenzione. La prossima volta, passa più tempo sul progetto del programma e sul disegno del diagramma di flusso e ne passerai molto meno nella ricerca degli errori.
Ora il tuo programma funziona correttamente, ritieni sia corretto e non vuoi toccarlo più. Comunque un errore potrebbe essere trovato in seguito, oppure potresti usare il programma di nuovo o unirlo con un altro. Per rendere un programma riutilizzabile, è richiesto ancora un passo: devi documentare il tuo programma per ricordare cosa fa.

Documentazione

Hai finito di progettare e codificare un programma; ti sei perfettamente impratichito con le sue operazioni e con quello che compiono le istruzioni. Bene, resterai sorpreso nello scoprire in quanto breve tempo dimenticherai cosa fa il programma e quanto è difficile, in seguito, rileggerlo e capirlo. Se hai l'intenzione di riutilizzare il programma o correggerne gli errori trovati in futuro, è vitale che tu spieghi il programma completamente ed in modo immediato.
Questo significa render chiaro il programma in se stesso cioè documentare ogni cosa che richiede una spiegazione, sulla carta o attraverso i REM nel programma.

Ecco di seguito un riassunto delle tecniche che abbiamo descritto per rendere un programma più chiaro:

Documenta i tuoi programmi.

Spiega cosa fai: Usa le istruzioni REM per spiegare formule, testi, nomi o convenzioni. È anche una buona idea di aggiungere una breve descrizione scritta di ogni metodo o tecnica che hai usato che non sia ovvia o non sia descritta con le istruzioni PRINT.

Metti in ordine i diagrammi di flusso: Presenta un diagramma di flusso o una serie di diagrammi di flusso in modo ordinato e che corrispondano esattamente al programma.

Spesso, durante il processo di ricerca degli errori, le ultime piccole variazioni sono fatte direttamente sul programma, perciò assicurati che esse siano riportate sul diagramma originale; altrimenti ti troverai in grosse difficoltà per effettuare in futuro cambiamenti o correzioni del programma.

Renumera le linee: Spesso in un processo di eliminazione degli errori, hai la necessità di inserire altre istruzioni. Quando reputi che il programma sia corretto, è una buona idea di renumerare tutte le linee in modo che tutte le linee siano spaziate regolarmente. Questo faciliterà ulteriori cambiamenti nel programma. Sono infatti disponibili alcuni speciali programmi di renumerazione di linee per renumerare un intero programma. Se un tale programma non è disponibile, è una buona idea di spendere un po' di tempo a renumerare le istruzioni per avere una sequenza più ordinata.

Usa una chiara impaginazione: Separa le sezioni con linee bianche o istruzioni vuote. Usa l'allineamento diversificato per maggior chiarezza. Puoi usare numeri di linea che hanno la stessa lunghezza per allineare verticalmente tutte le istruzioni. Inoltre, ogni volta che usi una istruzione FOR...NEXT, è una buona idea allineare più internamente tutto il blocco di istruzioni compreso tra il FOR ed il NEXT.
Inoltre usa gli spazi generosamente per rendere più chiare le istruzioni complesse, in particolare quelle che contengono una espressione matematica. Usa le parentesi per rendere più chiari i risultati dei calcoli.

Riassunto

In questo capitolo abbiamo descritto i cinque passi necessari per avere un programma finito: progetto, disegno del diagramma di flusso, codificazione, ricerca degli errori e documentazione. Rivediamoli.
Ogni programma richiede il progetto di un algoritmo.
Esso deve essere almeno studiato mentalmente, se non sulla carta. L'algoritmo può essere allungato con una serie di formule e di note che descrivono i passi essenziali.
Il passo seguente è il disegno del diagramma di flusso che descrive la completa sequenza degli eventi. Consideralo come un passo obbligatorio per ogni programma che sia composto da più di poche linee.

Ricordati che tanto più attentamente compilerai il diagramma di flusso, tanto maggiori saranno le probabilità che il tuo programma sia corretto.
Il passo seguente è la codificazione: il diagramma di flusso viene tradotto in istruzioni BASIC. La pratica accellererà notevolmente questa fase. Infatti la codificazione diventerà rapidamente la fase più breve.

Poi viene la fase di prova e ricerca errori. Questa fase è sempre richiesta e spesso è la più lunga. Ogni programma deve essere sempre attentamente controllato.
Per finire, la qualità della documentazione faciliterà o viceversa impedirà un uso futuro o una modifica del programma.

Esercizi

8.1: Descrivi le cinque fasi dello sviluppo di un programma.

8.2: Qual è la differenza tra codificazione e programmazione.

8.3: Quale lo scopo del debugging?

8.4: Come "tracci" una variabile? (tracing).

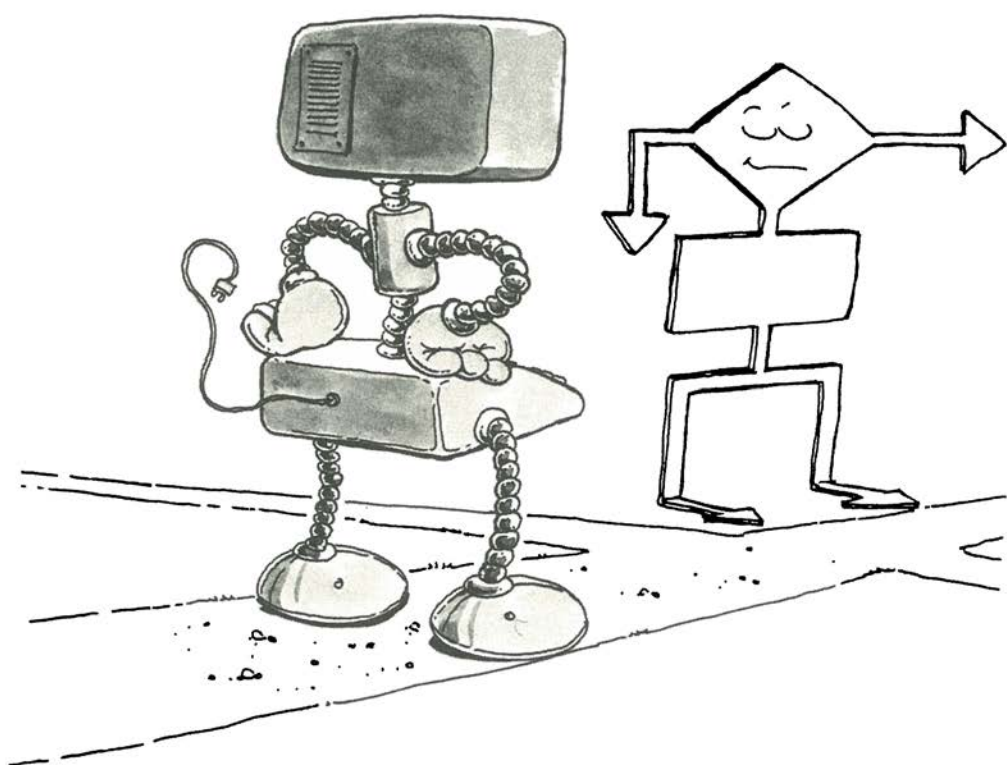
8.5: Perchè renumeri un programma dopo aver fatto molte variazioni?

8.6: Cosa è un diagramma di flusso?

8.7: Scrivi un diagramma di flusso per far partire la tua auto o per accendere un elettrodomestico?

8.8: Quali sono i vantaggi di un programma scritto in modo chiaro?

8.9 Descrivi le tecniche che possono essere usate per rendere più chiaro un programma?

Capitolo 9

Un esempio: conversione al sistema metrico

Svilupperemo ora un programma completo, descrivendolo un passo dopo l'altro. Ecco il problema da risolvere:
Devi scrivere un programma che converta automaticamente un peso espresso in once nel suo valore equivalente in grammi. Il programma dovrà sia convertire un numero introdotto dalla tastiera che visualizzare una tabella di conversione di pesi tra due valori specificati.

Progettiamo l'algoritmo

La sequenza approssimativa dei passi da seguire per risolvere questo problema è quasi immediata: dovremo far chiedere all'utente cosa desidera (una conversione singola o una tabella di valori) e poi eseguire l'azione richiesta.
Questo è il nostro algoritmo di massima. Rifiniamolo.
Un'oncia equivale a 28.35 grammi. La conversione da once a grammi è quindi eseguita dalla formula seguente:

P32anni = Ponze x 28.35

o in breve:

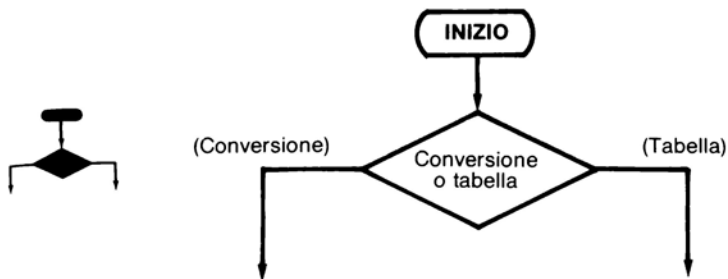
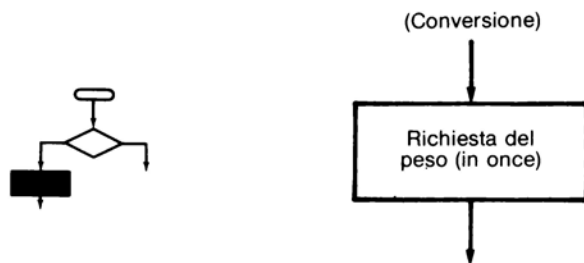
$P_g = P_{oz} \times 28.35$

Ecco l'algoritmo di base:

- ▸ Scelta fra singola conversione o tabella.
- ▸ Nel caso della conversione, richiedi il peso in once.
- ▸ Converti in grammi (usando la formula qui sopra) e visualizza il risultato.
- ▸ FINE.
- ▸ Nel caso della tabella, richiedi il massimo peso in once.
- ▸ Converti in grammi e mostra il risultato fino al limite.
- ▸ FINE.

In pratica non c'è alcun bisogno di scrivere l'algoritmo, purchè tu prepari il diagramma di flusso.

Diagramma di flusso

Nella preparazione del diagramma di flusso, dobbiamo prima sapere se l'utente desidera eseguire una conversione singola o visualizzare una tabella di valori. Ecco il corrispondente elemento del diagramma:

Questo è un elemento decisionale con due possibili uscite (ramificazioni): CONVERSIONE e TABELLA.
Esaminiamo prima la CONVERSIONE: l'utente desidera convertire un singolo peso dal valore in once all'equivalente in grammi. Dobbiamo richiedere, perciò, il valore del peso. Ecco la corrispondente parte del diagramma di flusso:

Potremmo, fin d'ora, convertire questo valore in grammi, comunque rifiniamo subito il diagramma di flusso. Come precauzione, controlliamo il valore introdotto dall'utente, verificando che sia un valore ragionevole. Ecco come diventa il nostro diagramma di flusso con l'aggiunta di questo controllo:

INIZIO

(Conversione)

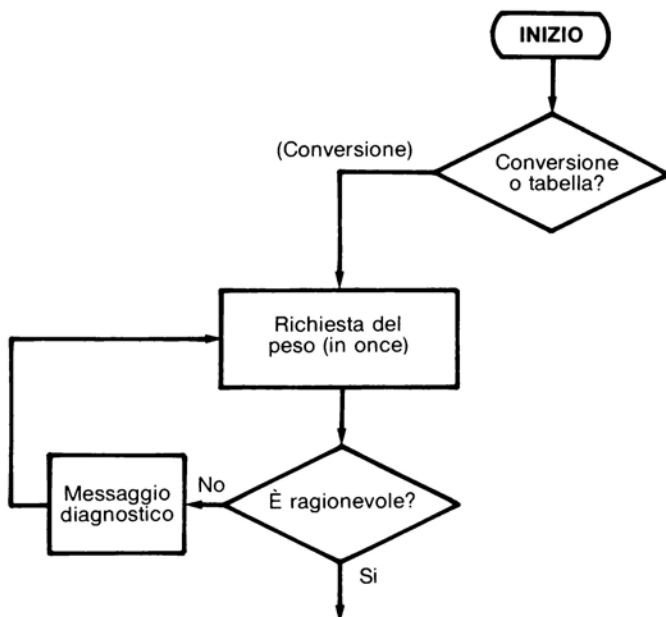
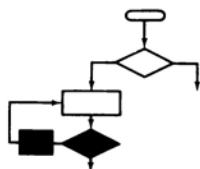
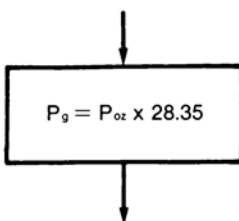
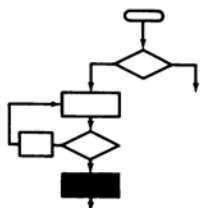
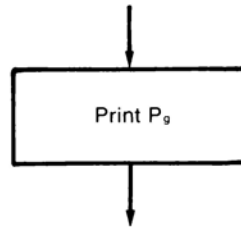
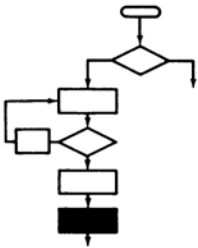
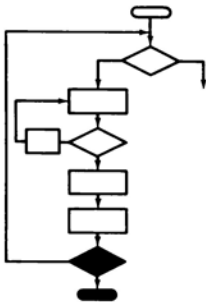
Conversione o tabella?

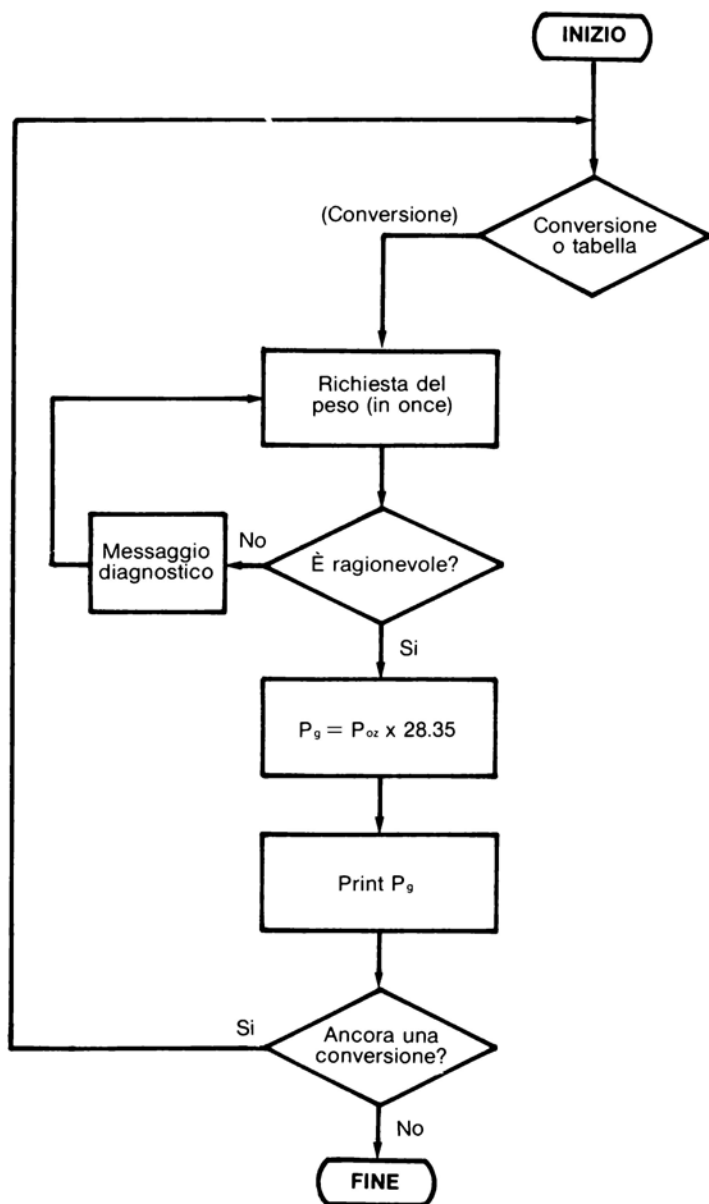
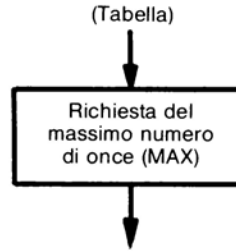
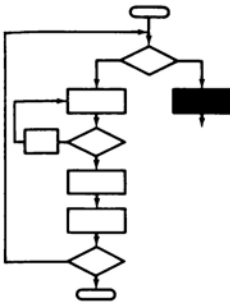
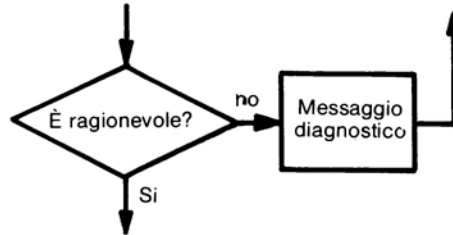
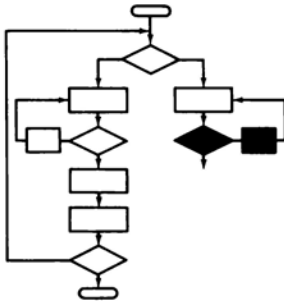
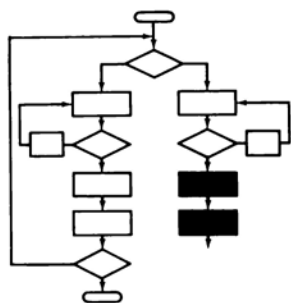
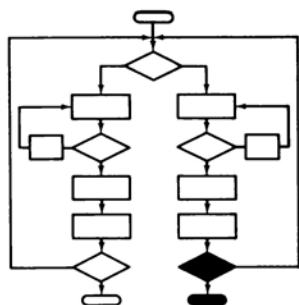
Richiesta del peso (in once)

Messaggio diagnostico

No

È ragionevole?

Si

Osserva che abbiamo aggiunto un elemento per verificare che il dato in ingresso sia ragionevole. In caso positivo, andiamo avanti; in caso contrario, viene inviato un messaggio diagnostico come "INPUT ASSURDO, PROVA DI NUOVO" e il programma richiede un altro valore del peso.
Ora che abbiamo controllato il valore del peso, convertiamolo in grammi. Questa azione è compiuta da:

Possiamo ora visualizzare il risultato. Ciò è fatto dal seguente elemento:

La conversione singola è finita e potremmo finire qui questa parte del diagramma di flusso, ma aggiungiamoci una conveniente possibilità: chiedere all'utente se vuole effettuare un'altra conversione. Ecco il relativo elemento:

Il simbolo di INIZIO sulla freccia sinistra indica che questa freccia verrà collegata dall'inizio del diagramma di flusso. A questo punto, il nostro diagramma di flusso si presenta così:

INIZIO
(Conversione)
Conversione o tabella
Richiesta del peso (in once)
Messaggio diagnostico
No
È ragionevole?
Si
$P_g = P_{oz} \times 28.35$
Print P_g
Si
Ancora una conversione?
No
FINE

Torniamo al nostro primo elemento di decisione ed esaminiamo cosa accade quando l'utente vuole visualizzare una tabella di valori. Questa è l'opzione TABELLA all'inizio del diagramma di flusso.
Dobbiamo conoscere il massimo valore da convertire.
Questa azione è compiuta dal seguente elemento:

Di nuovo, per sicurezza, controlliamo che questo numero sia ragionevole:

Come prima, il programma non va avanti, finchè l'utente non introdurrà un valore accettabile per MAX.
Una volta che è stato introdotto un valore accettabile, possiamo procedere a visualizzare una tabella che converta once in grammi fino al limite desiderato:

Per finire, aggiungiamo la stessa conveniente proprietà e chiediamo all'utente se voglia effettuare una nuova conversione:

La Figura 9.1 mostra il diagramma di flusso completo.
Questo diagramma di flusso è tipico. I contenuti dei suoi elementi sono scritti "di getto". Alcuni elementi saranno codificati con una o due istruzioni BASIC mentre altri ne richiederanno molte di più. Comunque è la sequenza ciò che conta, perchè il fatto che alcuni elementi siano scritti più dettagliatamente di altri, non ha alcuna importanza.

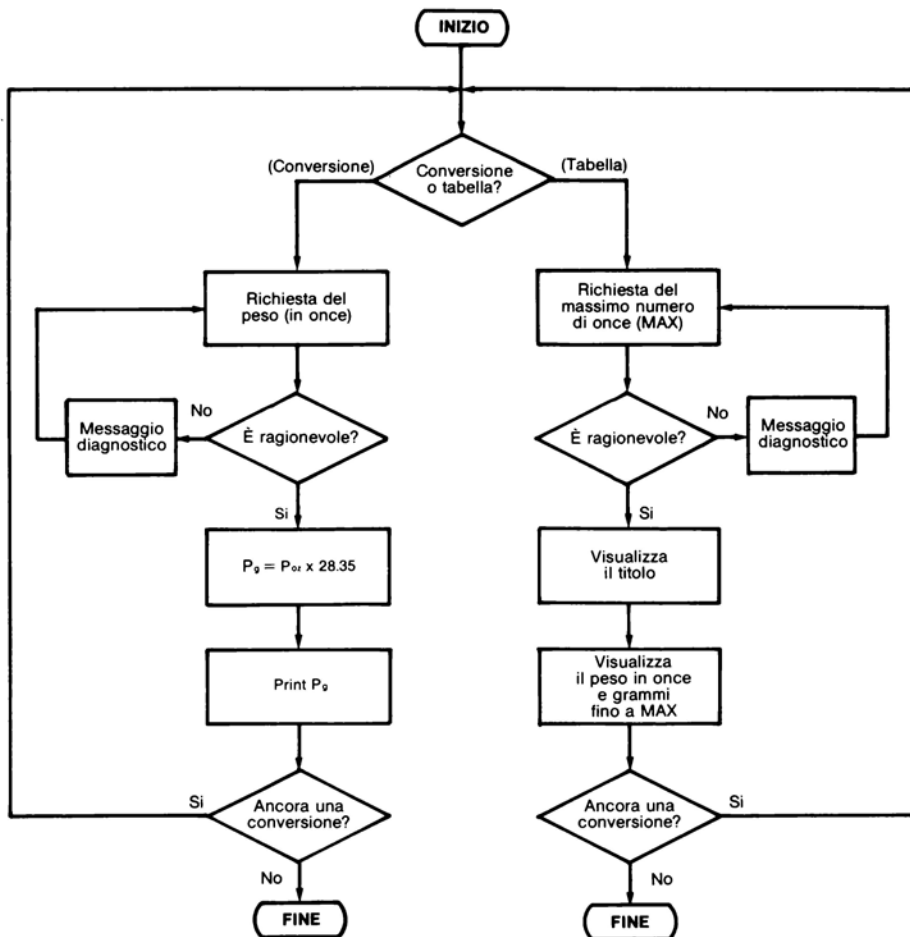



Figura 9.1 - Diagramma di flusso di conversione dei pesi.

Ricordati che la sequenza del diagramma di flusso deve essere esatta, ma i dettagli possono essere scritti nel modo che ti sembra più conveniente. Non c'è alcun bisogno di spendere ulteriore tempo per ottimizzare il contenuto degli elementi, quando ritieni di poterli codificare in modo semplice.

Il diagramma di flusso deve essere chiaro e facilmente leggibile. Un diagramma di flusso chiaro e ben organizzato migliora le tue possibilità di scrivere un programma corretto. Per completezza, ci sono alcune rifiniture o alternative che potresti prendere in considerazione:

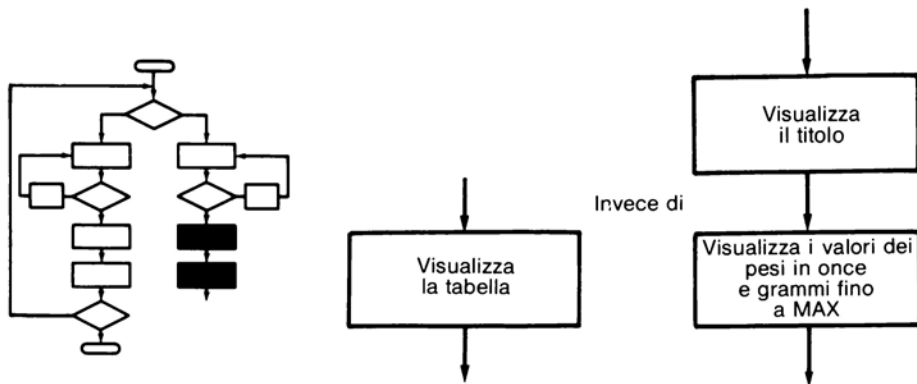
- Potresti spiegare nei dettagli, il modo di controllo di "ragionevolezza del valore". (Qui si controllerà semplicemente che il valore di POZ sia positivo).

- Potresti essere più esplicito sul messaggio diagnostico e sulla maggior parte del dialogo.

In generale, il consiglio è: *lascialo semplice.*
Perfezionalo quanto basta, perchè:

1. La sequenza dei passi sia corretta e completa.

2. Tu comprenda bene ogni elemento e sappia approssimativamente come convertirlo in istruzioni di programma.

Quanto più semplice è il contenuto di ogni elemento, tanto più chiaro sarà il diagramma di flusso. Tanto più dettagliato è il contenuto degli elementi, tanto più semplice sarà la codificazione. Applicando questo consiglio, potremmo semplificare il diagramma di flusso scrivendo:

Entrambe le opzioni sono corrette, perciò usa quella che ritieni più comoda. Qui ho deciso di prestare attenzione ai passi impliciti di programmazione e perciò ho reso più espliciti i contenuti degli elementi.
Puoi sempre riscrivere i contenuti degli elementi o per questo motivo, alcune parti del tuo diagramma di flusso su un foglio di carta separato per facilitare la codificazione o provare un'alternativa.
Ora che abbiamo un diagramma di flusso proviamo a mano con numeri attuali per vedere se funziona. Questo procedimento di prova manuale è banale, quindi andiamo avanti.

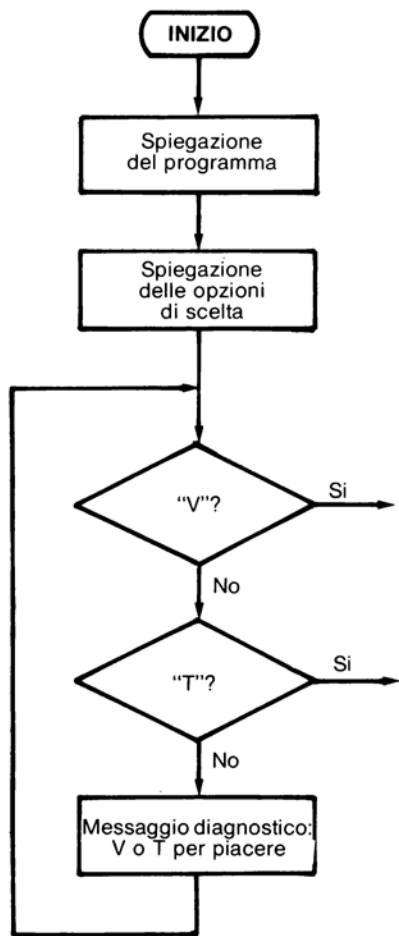
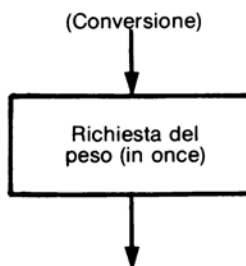
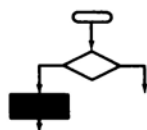
Scriveremo ora il programma che corrisponde al nostro diagramma di flusso. Per facilitarne la leggibilità, assumeremo che il tuo interprete BASIC permetta l'uso dei nomi di variabili lunghi (nomi che usano più di una lettera) e preveda l'uso delle operazioni a virgola mobile (cioè permetta l'uso dei numeri decimali).
Se questo non è il tuo caso, dovrai semplicemente abbreviare i nomi delle variabili e se il tuo è un Integer BASIC (permessi solo numeri interi), il programma funzionerà ugualmente, ma la conversione sarà approssimata.
Codifichiamo ora ogni elemento nelle corrispondenti istruzioni BASIC.

Ecco il primo elemento del diagramma di flusso:

ed ecco le corrispondenti istruzioni del programma:

```
100 REM * * * CONVERSIONE DA ONCE A GRAMMI * * *
110 REM QUESTO PROGRAMMA ESEGUE SIA UNA
    CONVERSIONE DIRETTA
120 REM CHE UNA STAMPA DI UNA TABELLA DI VALORI
130 REM PRIMA, SPECIFICA IL MODO DI CONVERSIONE:
    DIRETTO O TAVOLA
140 PRINT "CONVERTIRO' ONCE IN GRAMMI"
150 PRINT "SE VUOI CONVERTIRE UN VALORE, SCRIVI V"
160 PRINT "SE VUOI UNA TABELLA DI VALORI, SCRIVI T"
170 PRINT "QUALE SCEGLI [V O T]";
180 INPUT SCELTA$  .
190 IF SCELTA$ = "V" THEN GOTO 300
200 REM LA LINEA 300 INDICA LA CONVERSIONE DIRETTA
210 IF SCELTA$ = "T" THEN GOTO 500
220 REM LA LINEA 500 INDICA LA TABELLA DEI VALORI
230 REM SE IL CARATTERE NON ERA NE' V NE' T,
    L'INPUT NON E' VALIDO
240 PRINT "V O T, PER PIACERE: ";
250 GOTO 170
```

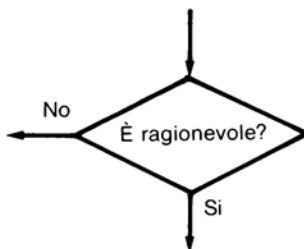
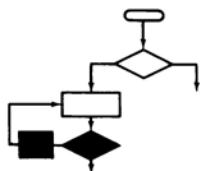
Quando avrai più esperienza, sarai in grado di passare direttamente dal primo elemento del diagramma di flusso alle corrispondenti istruzioni, ma, inizialmente, avrai bisogno di scriverne una versione dettagliata.

Ecco la versione dettagliata dell'elemento 1:

Osservando questa versione dettagliata, nota quanto sia vicino il diagramma di flusso alle istruzioni BASIC.
Inoltre, osserva che abbiamo introdotto un test di validità per SCELTA$ e non abbiamo semplicemente assunto che l'utente cooperasse introducendo "V" o "T". Provalo.
Ricorda: Ogni volta che richiedi un dato in ingresso, devi sempre verificarlo.
Il resto è più semplice; convertiamo l'elemento 2:

Le istruzioni corrispondenti sono:

```
300  REM * * * CONVERSIONE DI UN VALORE * * *
310  PRINT "SCRIVI IL PESO IN ONCE ...";
320  INPUT POZ
```

Semplificando, potremmo scrivere:

```
310  INPUT "SCRIVI IL PESO IN ONCE ...";POZ
```

Qui ho deciso di separare le istruzioni PRINT e INPUT per mostrare più facilmente la corrispondenza tra le linee del programma e gli elementi del diagramma di flusso, ma puoi comunque usare entrambe le forme.
Andiamo avanti con l'elemento 3: .

Ecco il suo programma equivalente:

```
330   IF POZ < 0 THEN GOTO 310
340   REM IL PESO DEVE ESSERE POSITIVO
350   REM POTREMMO AGGIUNGERE QUI UN MESSAGGIO ESPLICITO
```

L'equivalente dell'elemento 4 è:

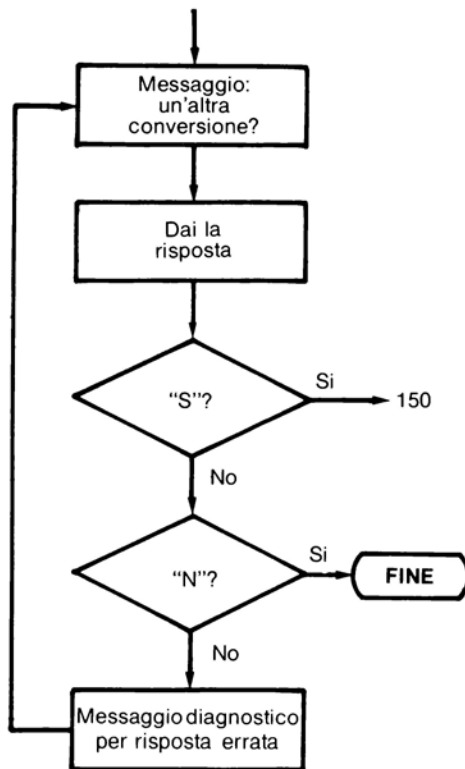
```
360   PG = POZ * 28.35
```

e per l'elemento 5:

```
370   PRINT POZ; "ONCE EQUIVALGONO A"; PG; "GRAMMI"
```

e per l'elemento 6:

```
410   PRINT "DESIDERI FARE ANCORA CONVERSIONI? S PER SI, N PER NO:";
420   INPUT NUOVO$
430   IF NUOVO$ = "S" THEN GOTO 150
440   IF NUOVO$ = "N" THEN END
450   REM L'INPUT NON ERA NE' S NE' N. DILLO ALL'UTENTE
460   PRINT "S O N, PER PIACERE"
470   GOTO 380
```

Se quanto sopra non ti è chiaro, ecco, in dettaglio, il diagramma di flusso equivalente:

Andiamo a guardare ora la parte destra del diagramma di flusso di Figura 9.1. Ecco l'equivalente dell'elemento 7:

```
520  PRINT "VISUALIZZERO' UNA TABELLA DI CONVERSIONE"
530  PRINT "DA ONCE A GRAMMI"
540  PRINT "DIMMI IL MASSIMO NUMERO DI ONCE:";
550  INPUT MAX
```

e per l'elemento 8:

```
560  REM MAX DEVE ESSERE MAGGIORE O UGUALE AD 1, ALTRIMENTI
     NON VERRA' ACCETTATO
570  IF MAX < 1 THEN GOTO 540
```

Per chiarezza, saltiamo una linea sullo schermo, prima di cominciare la tabella:

```
580  PRINT
```

Ricorda: Questa è una istruzione PRINT vuota che visualizza una linea di spazi. Ora ecco l'equivalente dell'elemento 9:

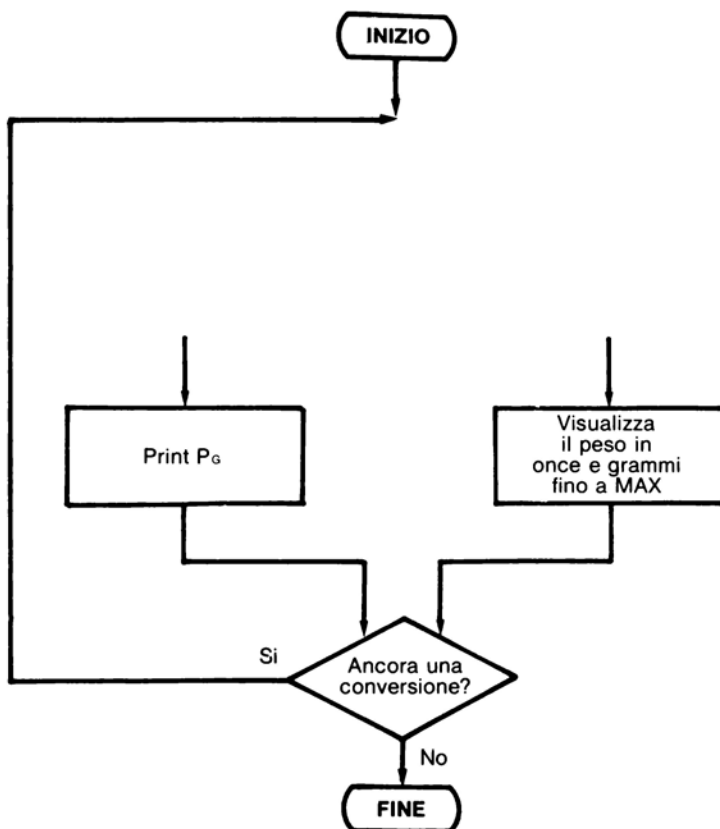
```
590  PRINT "ONCE", "GRAMMI"
```

Nota che abbiamo usato la virgola piuttosto che un punto e virgola per una buona spaziatura delle colonne.
Ed ecco l'elemento 10:

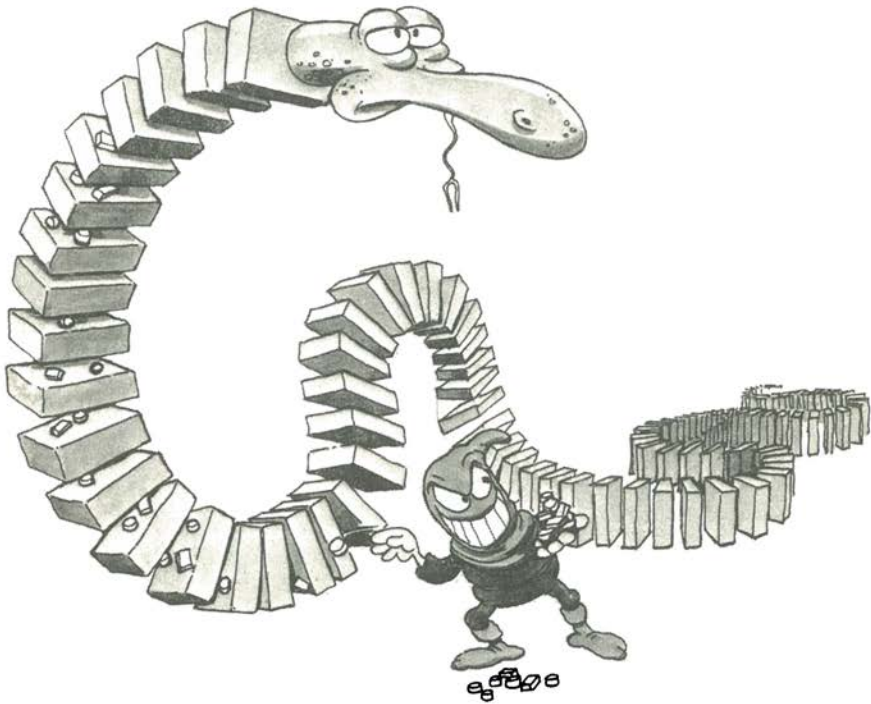
```
600  I = 1
610  PRINT I, I * 28.35
620  I = I + 1
630  IF I <= MAX GOTO 610
```

Per finire, ecco l'elemento 11:

```
650  GOTO 380
```

L'elemento 11 è uguale all'elemento 6 cosicchè possiamo semplificare il nostro programma con un salto all'elemento 6.
Ecco il corrispondente diagramma di flusso (corretto):

Il programma completo è mostrato alla pagina seguente:

```
100 REM * * * CONVERSIONE DA ONCE A GRAMMI * * *
110 REM QUESTO PROGRAMMA ESEGUE SIA UNA CONVERSIONE DIRETTA
120 REM CHE UNA STAMPA DI UNA TABELLA DI VALORI
130 REM PRIMA, SPECIFICA IL MODO DI CONVERSIONE: DIRETTO O TAVOLA
140 PRINT "CONVERTIRO' ONCE IN GRAMMI"
150 PRINT "SE VUOI CONVERTIRE UN VALORE, SCRIVI V"
160 PRINT "SE VUOI UNA TABELLA DI VALORI, SCRIVI T"
170 PRINT "QUALE SCEGLI [V O T]";
180 INPUT SCELTA$
190 IF SCELTA$ = "V" THEN GOTO 300
200 REM LA LINEA 300 INDICA LA CONVERSIONE DIRETTA
210 IF SCELTA$ = "T" THEN GOTO 500
220 REM LA LINEA 500 INDICA LA TABELLA DEI VALORI
230 REM SE IL CARATTERE NON ERA NE' V NE' T, L'INPUT NON E' VALIDO
240 PRINT "V O T, PER PIACERE: ";
250 GOTO 170
260 REM
270 REM
300 REM * * * CONVERSIONE DI UN VALORE * * *
310 PRINT "SCRIVI IL PESO IN ONCE...";
320 INPUT POZ
330 IF POZ < 0 THEN GOTO 310
340 REM IL PESO DEVE ESSERE POSITIVO
350 REM POTREMMO AGGIUNGERE QUI UN MESSAGGIO ESPLICITO
360 PG = POZ * 28.35
370 PRINT POZ; "ONCE EQUIVALGONO A"; PG; "GRAMMI"
380 REM
390 REM * * * MODULO DI USCITA * * *
400 PRINT
410 PRINT "DESIDERI FARE ANCORA CONVERSIONI? S PER SI, N PER NO:";
420 INPUT NUOVO$
430 IF NUOVO$ = "S" THEN GOTO 150
440 IF NUOVO$ = "N" THEN END
450 REM L'INPUT NON ERA NE' S NE' N, DILLO ALL'UTENTE
460 PRINT "S O N, PER PIACERE"
470 GOTO 380
480 REM
490 REM
500 REM * * * CONVERSIONE DI UNA TABELLA * * *
510 REM RICHIESTA DEL LIMITE SUPERIORE
520 PRINT "VISUALIZZERO' UNA TABELLA DI CONVERSIONE"
530 PRINT "DA ONCE A GRAMMI"
540 PRINT "DIMMI IL MASSIMO NUMERO DI ONCE:";
550 INPUT MAX
560 REM MAX DEVE ESSERE MAGGIORE O UGUALE AD 1, ALTRIMENTI
    NON E' ACCETTATO
570 IF MAX < 1 THEN GOTO 540
580 PRINT
590 PRINT "ONCE", "GRAMMI"
600 I = 1
610 PRINT I, I * 28.35
620 I = I + 1
630 IF I <= MAX GOTO 610
640 REM I E' ORA MAGGIORE DI MAX. QUESTA E' LA FINE DELLA TABELLA
650 GOTO 380
660 END
```

Ricorda: ogni cambiamento che fai, può introdurre nuovi errori!

Sono stati aggiunti alcuni miglioramenti. Per esempio, sono state aggiunte per chiarezza alcune istruzioni REM (vedi linee 260, 270, 300, 380, 390, 480, 490, 500).
Potrebbero essere fatti altri miglioramenti. Per esempio, le istruzioni da 600 a 630 potrebbero essere sostituite con un ciclo FOR...NEXT. Questo migliorerebbe la leggibilità, ma non ne vale molto la pena.
Ricorda: ogni cambiamento che operi su un programma che funziona, può introdurre nuovi errori (bug); perciò cambia il tuo programma solamente se c'è un chiaro vantaggio.
Abbiamo ora un programma completo. Proviamolo.

Collaudo

Eseguiamo il RUN ed ecco lo schermo:

```
RUN
CONVERTIRO' ONCE IN GRAMMI
SE VUOI CONVERTIRE UN VALORE, SCRIVI V.
SE VUOI UNA TABELLA DI VALORI, SCRIVI T.
QUALE SCEGLI [V O T]? V
SCRIVI IL PESO IN ONCE...? 3
3 ONCE EQUIVALGONO A 85.05 GRAMMI
```

ed eccone un altro:

```
DESIDERI FARE ANCORA CONVERSIONI? S PER SI,
N PER NO:? S
SE VUOI CONVERTIRE UN VALORE, SCRIVI V
SE VUOI UNA TABELLA DI VALORI, SCRIVI T
QUALE SCEGLI [V O T]? T
VISUALIZZERO' UNA TABELLA DI CONVERSIONE
DA ONCE A GRAMMI
DIMMI IL MASSIMO NUMERO DI ONCE:? 4

ONCE          GRAMMI
1             28.35
2             56.7
3             85.05
4             113.4
```

Il nostro programma sembra che funzioni tanto per una singola conversione quanto per una tabella di valori.

Proviamo ad ingannarlo:

```
DESIDERI FARE ANCORA CONVERSIONI? S PER SI,
N PER NO: ? S
SE VUOI CONVERTIRE UN VALORE, SCRIVI V
SE VUOI UN TABELLA DI VALORI, SCRIVI T
QUALE SCEGLI [V O T]? D
V O T PER PIACERE: QUALE SCEGLI [V O T] ?V
SCRIVI IL PESO IN ONCE...? 7
7 ONCE EQUIVALGONO A 198.45 GRAMMI
```

Proviamo l'opzione di ripetizione:

```
DESIDERI FARE ANCORA CONVERSIONI? S PER SI,
N PER NO: ? S
SE VUOI CONVERTIRE UN VALORE, SCRIVI V
SE VUOI UN TABELLA DI VALORI, SCRIVI T
QUALE SCEGLI [V O T]? V
SCRIVI IL PESO IN ONCE...? 85
85 ONCE EQUIVALGONO A 2409.75 GRAMMI


DESIDERI FARE ANCORA CONVERSIONI? S PER SI,
N PER NO: ? S
SE VUOI CONVERTIRE UN VALORE, SCRIVI V
SE VUOI UN TABELLA DI VALORI, SCRIVI T
QUALE SCEGLI [V O T]? V
SCRIVI IL PESO IN ONCE...? 2.5
2.5 ONCE EQUIVALGONO A 70.875 GRAMMI


DESIDERI FARE ANCORA CONVERSIONI? S PER SI,
N PER NO: ? S
SE VUOI CONVERTIRE UN VALORE, SCRIVI V
SE VUOI UN TABELLA DI VALORI, SCRIVI T
QUALE SCEGLI [V O T]? V
SCRIVI IL PESO IN ONCE...? 3.1
3.1 ONCE EQUIVALGONO A 87.885 GRAMMI
```

Proviamo ad ingannarlo di nuovo:

```
DESIDERI FARE ANCORA CONVERSIONI? S PER SI,
N PER NO: ? S
SE VUOI CONVERTIRE UN VALORE, SCRIVI V
SE VUOI UN TABELLA DI VALORI, SCRIVI T
QUALE SCEGLI [V O T]? V
SCRIVI IL PESO IN ONCE...? -5
SCRIVI IL PESO IN ONCE...?
```

Bene, sembra che funzioni, ma dovresti provarlo realmente molte altre volte, prima di essere completamente soddisfatto.
In questo caso, abbiamo fatto molta attenzione e siamo stati molto fortunati. Il nostro programma ha funzionato bene al primo tentativo.

Riassunto

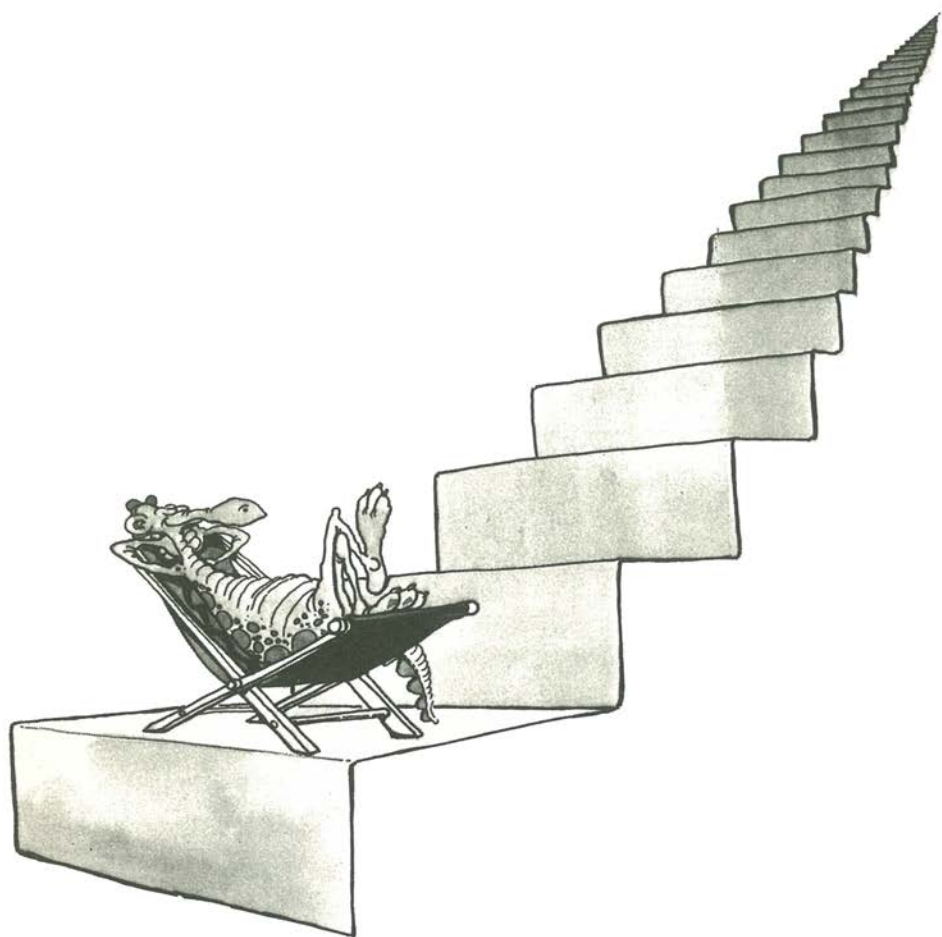
In questo capitolo abbiamo illustrato la sequenza completa necessaria per scrivere un programma che risolva un certo problema. Dovresti ora chiudere questo libro, scrivere un tuo diagramma di flusso e convertirlo in un programma funzionante.
Il segreto di una programmazione con successo è la pratica.

Esercizi

9.1: Aggiungi a questo programma l'opzione di convertire grammi in once.

9.2: Espandi il programma per includere la conversione delle distanze.
1 metro = 39.37079 pollici
1 km = 0.62138 miglia
1 pollice = 25.3995 mm
1 piede = 30.479 cm
1 yard = 0.91438 m
1 miglio = 1609.3149 m

9.3: Espandi il programma per includere la conversione di temperatura:
C = (F—32)x5/9
F = (9/5)xC-32

9.4: Suggerisci altri modi per migliorare o rendere più chiaro il nostro programma finale.

Capitolo 10

Il prossimo passo

Ora, hai imparato a scrivere i tuoi programmi in BASIC.
In questo capitolo esamineremo il prossimo passo che dovrai fare per migliorare la tua abilità di programmatore.
Guarderemo di nuovo cosa puoi fare con il BASIC e descriveremo poi tecniche ed argomenti pratici che possono aiutarti a scrivere più facilmente i programmi complessi.

Cosa puoi fare con il BASIC

Puoi scrivere un programma in BASIC per automatizzare la maggior parte dei lavori a meno che essi non richiedano calcoli matematici molto precisi, non debbano prendere complesse decisioni o non debbano fornire una risposta velocissima (come un controllo di processo in tempo reale).
Troverai che il BASIC si adatta bene a semplici applicazioni di lavoro come elaborazione di dati, indirizzari ed comuni calcoli finanziari. Con le estensioni del linguaggio, il BASIC si presta abbastanza bene alla grafica ed ai videogames.
Altre aree di applicazioni tipiche del BASIC includono l'educazione con l'aiuto del computer, l'archiviazione di dati personali e di lavoro, i calcoli matematici e tecnici con un limitato grado di precisione e molti altri impieghi.
Generalmente, le applicazioni sono principalmente limitate dall'esperienza del programmatore.
Con la conoscenza che hai acquisito finora, potrai scrivere un'ampia varietà di programmi in BASIC. Comunque, andando avanti, vorrai migliorare la tua capacità ed usare strumenti di programmazione più utili e più potenti.
Questo è l'argomento del prossimo paragrafo.

Migliorare la tua capacità

Puoi fare tre passi essenziali per migliorare la tua capacità in BASIC:

1. Fare molta pratica.

2. Avere una miglior conoscenza di tutte le risorse della tua versione di BASIC.

3. Imparare nuove tecniche di programmazione.

Vediamo questi passi uno dopo l'altro.

Fai più pratica.

Molta pratica

Il segreto di una efficace programmazione è la pratica: scrivi quanti più programmi ti è possibile e falli funzionare. Migliora l'aspetto della tua programmazione seguendo tutte le raccomandazioni presentate in questo libro. Se i tuoi programmi funzionano bene dopo pochi tentativi, puoi essere sulla strada buona per diventare un programmatore efficace e disciplinato. In caso contrario, riguarda il tuo comportamento o magari rileggi

alcune parti di questo libro e fai più pratica. **Ricorda**: non c'è altro modo di diventare un buon programmatore, se non di scrivere molti programmi. Questo libro ti permetterà di iniziare, ma niente può sostituire l'esperienza personale.

Migliora le tue capacità.

Caratteristiche specifiche del tuo BASIC

Ogni interprete BASIC è fornito di speciali capacità come istruzioni, comandi, abbreviazioni ed estensioni al "BASIC standard" (come grafica e suono) e di un proprío ambiente operativo (con comandi per la memorizzazione su dischi e cassette, strumenti per operare su file, un programma "editor" per la correzione dei testi" ed altro).
Puoi aumentare la tua abilità studiando queste caratteristiche ed agevolazioni. Nel prossimo paragrafo, descriveremo alcune istruzioni addizionali presenti nella maggior parte degli interpreti. Altre agevolazioni come grafica, suono e files, sono specifiche del tuo interprete e del tuo computer. Ti servirà molto leggere qualcosa su di esse.

Tecniche addizionali

Quando svilupperai programmi più lunghi (diciamo, più lunghi di una pagina), desidererai imparare le tecniche usuali per risolvere problemi comuni, come l'ordinamento e la classificazione di articoli, la formattazione di dati, le operazioni sui file e la creazione di strutture di dati.
Questi argomenti si trovano nei vari libri di programmazione.

Ancora BASIC

Ogni interprete BASIC offre una serie abbastanza "standardizzata" di caratteristiche e, in più, molte estensioni che sono specifiche dell'interprete. Le caratteristiche comuni, offerte dalla maggior parte dei BASIC, includono tutto quello che abbiamo studiato finora, più sei tipi di istruzioni. Presenteremo una breve esposizione di ognuno di questi tipi addizionali, perchè forse vorrai studiarli per conto tuo o con l'aiuto di un libro più avanzato. Essi sono:
1. funzioni: Le funzioni sono delle espressioni *predefinite* o *definite dall'utente* che operano su una determinata variabile ed eseguono un calcolo o una azione specifica. Il BASIC è fornito di alcune funzioni predefinite (come ABS, COS, EXP, INT, FIX, RND, SGN, SQR o TAN). Queste funzioni eseguiranno automaticamente compiti comuni. Altre funzioni possono essere definite dall'utente.
Esaminiamo ora alcune funzioni predefinite:

- La funzione INT calcola la parte intera di un numero decimale, troncando la parte frazionaria del numero. Per esempio INT (1.234) dà il valore 1.

- Similmente, ABS calcola il valore assoluto. Per esempio, ABS (—5.2) è 5.

- La funzione SQR calcola la radice quadrata di un numero. Per esempio, SQR (4) dà il valore 2.

Le funzioni possono anche essere definite dall'utente. Una funzione definita dall'utente è essenzialmente una formula, (scritta dall'utente), che ha un nome, opera su una variabile e può essere usata molte volte nello stesso programma. Cioè ogni volta che viene richiamato quel nome, la formula è calcolata con il valore corrente della variabile. È una convenzione abbreviata molto utile.
Ecco un esempio di una funzione definita dall'utente che calcola il 2% di X:

```
10  DEF FNA(X) = X * 2/100
```

ed ecco un modo di usare questa funzione:

20 PRINT FNA (50)

Questa istruzione visualizzerà il valore 1.
Analizziamola più da vicino. FN significa funzione e FNA è la funzione A, cioè il nome della funzione. X è una variabile "fittizia", cioè X non deve avere assegnato alcun valore quando viene scritta la DEFinizione. Il valore attuale o la variabile viene sostituito alla X quando la funzione viene usata; per esempio, si potrebbe scrivere FNA (50).

2. Sottoprogrammi (subroutine): Un sottoprogramma è un gruppo di istruzioni dentro un programma BASIC che ha un suo nome e che può essere ripetuto semplicemente scrivendone il nome. Perciò ogni volta che è usato nel corso del programma il nome del sottoprogramma, tutte le istruzioni della subroutine verranno eseguite. I sottoprogrammi sono utili per eseguire ripetutamente un segmento di programma senza dover ripetere le istruzioni ogni volta che sono richieste.
Ecco un esempio di sottoprogramma:

```
500 REM QUESTO UN SOTTOPROGRAMMA PER IL CALCOLO
    DELLA MEDIA
510 PRINT "CALCOLERO' LA MEDIA FRA A E B"
520 PRINT"A = ";A; "B = "; B
530 MEDIA = (A+B)/2
540 PRINT "LA MEDIA E' "; MEDIA
550 RETURN
```

Ammesso che A e B abbiamo un valore già assegnato, questo sottoprogramma può essere richiamato con un'istruzione del tipo:

50 GOSUB 500

Esso può essere usato di nuovo, in seguito, nello stesso programma con un'istruzione come:

170 GOSUB 500

e darà un valore diverso se A e B sono diversi. L'uso dei sottoprogrammi renderà i tuoi programmi più chiari, più corti e ti risparmierà molto tempo. Puoi anche costituire un archivio dei sottoprogrammi più comuni ed usarli in vari programmi. Comunque stai attento ai nomi delle variabili ed ai numeri delle linee!

3. Operatori su stringhe: Gli operatori su stringhe ti permettono di manipolare convenientemente i testi operando su stringhe di caratteri. Gli operatori su stringhe possono servire per modificarle, misurarle, congiungerle con altre, dividerle, inserire un testo a destra, a sinistra o in una data posizione, sostituire i caratteri o far confronti con altre

stringhe. Questi operatori sono utili per elaborazioni su parole o su testi.

4. Strutture di dati: Il BASIC permette l'uso delle variabili indicizzate ad uno o due indici. Queste variabili corrispondono, in matematica, ai vettori e alle matrici (array). L'uso delle variabili indicizzate ti permette la costruzione di strutture come liste e la possibilità di riferirsi ad ogni elemento di esse. Per esempio, ci si può riferire agli elementi di una lista di nome CLIENTE con CLIENTE(1), CLIENTE(2), ecc. e puoi visualizzare tutti e dieci i valori scrivendo:

```
50  FOR N=1 TO 10
60  PRINT CLIENTE(N)
70  NEXT N
```

o puoi confrontare due elementi consecutivi con:

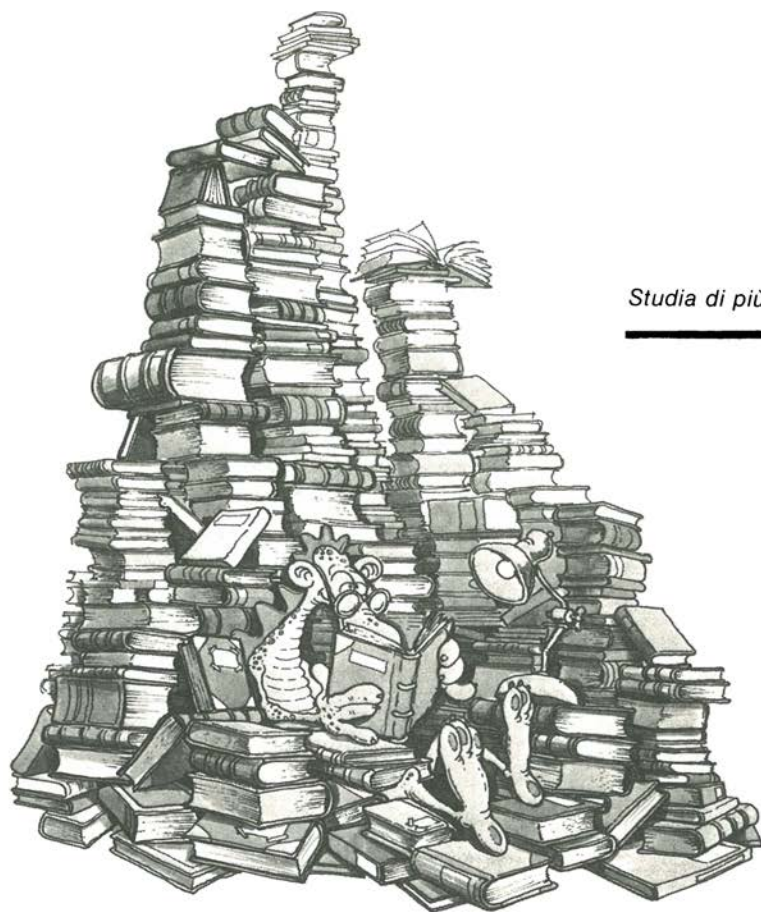
```
100  IF CLIENTE(K) v CLIENTE(K+) THEN 500
```

Questo è un grosso vantaggio.

5. Files: Ogni disk-BASIC è fornito di istruzioni per la memorizzazione ed il recupero di dati su disco. Queste istruzioni sono specifiche del tuo computer e del tuo interprete e sono descritte nella documentazione del costruttore.

6. Risorse addizionali: Con un buon BASIC sono generalmente disponibili anche altre risorse come le seguenti istruzioni: ON...GOTO, READ...DATA e PRINT USING.
Potrai esplorare queste facilitazioni, che rendono la programmazione più semplice, attraverso il manuale del tuo interprete o con un altro libro.
Possono essere disponibili anche delle abbreviazioni; per esempio, potresti poter scrivere:

```
? 25 * 32
```

ecco un utile comando per usare il tuo computer come calcolatrice tascabile.
Inoltre, potrebbe essere disponibile un modo a doppia precisione per aumentare la precisione dei calcoli numerici e potrebbe essere disponibile una istruzione di tabulazione (TAB) per facilitare la visualizzazione delle tabelle.
Per finire, ricorda che ci sono dei comandi specifici del tuo interprete per generare grafici, produrre effetti sonori e facilitare l'editing (per esempio, comandi che ti permettono di modificare i programi ed i files creati), aiutarti nell'esecuzione e nella ricerca degli errori sui tuoi programmi. Tali agevolazioni includono il controllo dello schermo, la disposizione lungo il programma di punti di arresto (dove il programma si fermerà automaticamente), la traccia (durante l'esecuzione) dei valori di variabili selezionate e l'incremento o la diminuzione della velocità di visualizzazione.

Studia di più.

Conclusione

Lo scopo di questo libro è stato di insegnarti, rapidamente ed efficacemente, a scrivere i tuoi primi programmi in BASIC. Se ora sei veramente interessato al BASIC, vorrai saperne di più, perciò il tuo prossimo passo dovrebbe essere quello di lavorare su tutti gli esercizi e di sviluppare alcuni programmi tuoi. Progredendo, vorrai impararne di più sulle tecniche più avanzate e per questo alla fine del libro viene presentata una lista di libri sulla programmazione.
Spero converrai che imparare il BASIC può essere facile e divertente e spero che il tuo desiderio sia di saperne di più. Gradirei sapere da te se hai dei suggerimenti per i possibili miglioramenti di questo libro.

Appendice **A**

Risposte ad alcuni esercizi

2

2.2:

```
10 PRINT "AAAAA"
20 PRINT "BBBB"
30 PRINT "CCC"
40 PRINT "DD"
50 PRINT "E"
```

2.4: **a.** Nel BASIC, un'etichetta è un numero di linea e deve sempre precedere ogni istruzione che fa parte di un programma.

b. Il modo ad esecuzione differita è il modo normale nel quale è introdotto un programma. Le istruzioni BASIC sono scritte con i numeri di linea e vengono memorizzate dal calcolatore per una esecuzione seguente.

c. L'esecuzione immediata è il modo nel quale una istruzione è scritta senza numero di linea e viene eseguita immediatamente. Questo è chiamato anche modo di calcolo.

d. Una istruzione vuota è una istruzione che non fa nulla. Tipicamente è un numero di linea o etichetta da sola (cioè senza nient'altro sulla stessa linea) o con "REM" .

e. Un cursore è uno speciale simbolo visivo sullo schermo (un quadratino o una lineetta) che mostra la tua posizione corrente. Generalmente lampeggia per migliorare la visibilità.

f. Il tasto di controllo o CTRL, quando viene premuto insieme con un tasto alfabetico, introduce un comando specifico. I tasti di controllo rendono più semplice inviare al computer i comandi più frequentemente usati, perchè devono venire premuti solamente due tasti.

g. Un key pad è una tastiera. Possiede generalmente una piccola serie di caratteri per uso speciale ed è posizionato alla destra della tastiera principale. È usato normalmente per i calcoli numerici e per il movimento del cursore.
h. Una parola riservata è un nome che ha uno specifico significato per il BASIC. Non può essere usata dal programmatore come nome di variabile.
i. Il prompt è un carattere o un messaggio generato da un programma che indica che il programma attende che l'utente introduca una informazione. La maggior parte dei BASIC usano un prompt, come ">", che significa: "scrivi la prossima istruzione". Il prompt "?" significa "introduci un dato".

2.6: Sì, ma è un metodo molto scomodo per farlo, perchè se desidererai eseguire il programma di nuovo, dovrai riscrivere ancora tutte le istruzioni. Anche i salti condizionali (che verranno trattati in seguito) non funzionano correttamente se scritti in questo modo.

2.8: Si, il BASIC inserirà ogni istruzione al suo posto nel programma così da avere tutte le etichette in ordine crescente.

2.10: No, devi scrivere:

```
PRINT "ESEMPIO"
```

2.12: Per cancellare l'istruzione 20 in un programma, scrivi una istruzione vuota con l'etichetta 20.

3

3.2:

```
PRINT 1 + (1/2) * (1/(1 + (1/2)))
```

o:

```
PRINT 1 + .5/(1 + .5)
```

3.4:

```
PRINT 100 * 1.6
```

3.6:

```
PRINT 350/55
```

4

4.1:

```
10 INPUT A, B, C, D
20 SOMMA = A + B + C + D
30 MEDIA = SOMMA/4
40 PRODOTTO = A * B * C * D
50 PRINT SOMMA, MEDIA, PRODOTTO
60 END
```

4.2:

a - no   e - si   i - si
b - si   f - si   j - no
c - no   g - no   k - si
d - si   h - no   l - si

4.4:

```
10 PRINT "DAMMI IL NOME DI UN OGGETTO ";
20 INPUT O$
30 PRINT "DAMMI IL NOME DI UN MOBILE ";
40 INPUT M$
50 PRINT "DAMMI IL NOME DI UN AMICO ";
60 INPUT A$
70 PRINT
80 PRINT "IL TUO AMICO "; A$;" HA UN "; O$;
   " SOPRA UN "; M$; " ?"
90 END
```

4.6:

b, c, f sono esatte.

5

5.2:

a - si   d - si
b - si   e - no
c - si   f - no

5.4:

```
10 INPUT "IL TUO NOME: "; NOME$
20 INPUT "QUANTO FA 2 + 3?"; C
30 INPUT "GLI ULTIMI DUE NUMERI SONO..."; A,B
```

5.6:

A = 3

6.1: L'istruzione IF permette al programma di prendere decisioni, perciò cambia il suo comportamento a seconda delle variazioni dei dati in ingresso o dei valori calcolati.

6.3:

a - si	e - si
b - si	f - si
c - si	g - si
d - si	

6.4: Si

6.5: Un ciclo esegue ripetutamente una parte del programma. Per evitare la creazione di cicli infiniti (un ciclo che non si arresta mai), bisognerebbe sempre effettuare un confronto dentro il ciclo, che, se vero, permetta di uscir fuori dal ciclo stesso.

7

7.2:

```
10  INPUT "ORE, MINUTI :"; ORE, MINUTI
20  REM CONTROLLA I DATI
30  IF ORE >= 0 AND ORE < 72
    AND MINUTI >= 0 AND MINUTI < 60 THEN 70
40  PRINT "DATI NON CORRETTI, PROVA DI NUOVO"
50  GOTO 10
60  REM STAMPA LINEA DI H
```

```
70  IF ORE = 0 THEN 110
80  FOR A = 1 TO ORE
90  PRINT "H";
100 NEXT A
110 PRINT
120 REM STAMPA LINEA DI M
130 IF MINUTI = 0 THEN 170
140 FOR A = 1 TO MINUTI
150 PRINT "M";
160 NEXT A
170 PRINT
180 END
```

7.4: Un salto può essere fatto dentro un ciclo non eseguito tramite l'istruzione FOR...NEXT.

7.6:

```
10  PRINT "PROGRAMMA PER SOMMARE TUTTI I NUMERI
    INTERI DISPARI"
20  PRINT "FINO AL LIMITE IMPOSTO DALL'UTENTE"
30  INPUT "VALORE NUMERO DISPARI PIU' ALTO DA SOMMARE
    "; NUM
40  REM TEST PER INPUT VALIDO
50  REM INT(NUM/2) <>  NUM/2 VALUTA SE IL NUMERO
    E' DISPARI - VEDI CAP. 10
60  IF NUM > 0 AND NUM < 1000 AND INT(NUM/2) <
    NUM/2 THEN 90
70  PRINT "NUMERO NON VALIDO...PROVA DI NUOVO"
80  GOTO 30
90  REM SCRIVI LA TABELLA
100 PRINT "NUMERO", "SOMMA"
110 PRINT
120 FOR N = 1 TO NUM STEP 2
130 SOMMA = SOMMA + N
140 PRINT N, SOMMA
150 NEXT N
160 END
```

7.8:

```
10 INPUT "PERCENTUALE IVA "; IVA
20 IF IVA < 1 OR IVA > 100 THEN 10
30 PRINT "PREZZO", "IVA", "PREZZO + IVA"
40 FOR PREZZO = 1 TO 100
50 PRINT PREZZO, PREZZO * IVA/100, PREZZO + PREZZO *
   IVA/100
60 NEXT PREZZO
70 END
```

8.2: La codificazione è un passo della programmazione. La programmazione include il progetto dell'algoritmo, il disegno del diagramma di flusso, la codificazione, la ricerca degli errori e la documentazione.

8.4: Una variabile viene "tracciata" inserendo delle istruzioni PRINT per mostrare quali valori assume la variabile nei punti critici del programma. Qualche volta il comando TRACE viene fornito dall'interprete per facilitare questa operazione.

8.6: Un diagramma di flusso è un diagramma simbolico che mostra la sequenza degli eventi che si manifestano durante l'esecuzione del programma.

8.8: I programmi scritti in modo chiaro sono facili da capire e perciò facili da modificare. Ciò permette al programmatore che ha creato il programma, così come ad altri programmatori, di effettuare facilmente delle variazioni.

Appendice **B**

Le più comuni parole riservate del BASIC

Questa lista ti aiuterà ad evitare l'uso di nomi di variabili non permessi anche se alcune di queste parole, probabilmente, non sono riservate dal tuo interprete.

ABS
AND
ARCCOS
ARCSIN
ARCTAN
AT
AUTO
BREAK
CALL
CHR$
CLEAR
CLOCK
CLOSE
CLS
COLOR
CON
CONT
COPY
COS
DATA
DATE
DEFDBL
DEFFN
DEFOMT
DEFSNG
DEFSTR
DEFUSR
DELETE
DIM
DSP
EDIT
ELSE
END
ENTER
EOF
ERR
ERROR
EXP
FIELD
FIX
FN
FOR
FORMAT
FREE
FUNCTION
GET
GOSUB
GOTO
GRAPHICS
HUN
IF
INKEY$
INP
INPUT
INSTR
INT
KILL
LEFT$
LET
LSET
LEN
LINE
LIST
LN
LOAD
LOC
LOG
LPRINT
MEM
MERGE
MID$
MKD$
MKI$
MKS$
NEW
NEXT
NOT
ON
OPEN
OR
OUT
PAUSE
PEEK
PLOT
POINT
POKE
POP
POS
POSN
POWER
PRINT
PUT
RANDOM
READ
REM
RENAME
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RUN
SAVE
SET
SGN
SIN
SLOW
SQR
STEP
STOP
STRING$
STR$
TAB
TAN
TEXT
THEN
TIME$
TO
TRACE
TROFF
TRON
UNPLOT
USING
USR
VAL
VARPTR
VERIFY
VLIN
VTAB

Glossario del BASIC

alfanumerico L'insieme dei caratteri alfabetici e numerici.

algoritmo Una sequenza di passi che definisce la soluzione di un dato problema.

assegnazione L'operazione di dare un valore ad una variabile, che è indicata nel BASIC con il simbolo =.

BASIC *Beginners All-purpose Simbolic Instruction Code* (Codice ad istruzioni simboliche per principianti adatto a tutti gli usi). Un linguaggio programmativo ad alto livello progettato per essere appreso con facilità.

binario Un sistema di numerazione che usa soltanto due cifre: 0 e 1.

bit Una contrazione delle parole "BInary digiT" (cifra binaria). Un bit può assumere il valore 0 o 1.

bug Errore di programma. I bug dovrebbero essere prevenuti piuttosto che curati.

byte Un gruppo di otto bit.

caricamento Trasferimento di dati o di un programma nella memoria interna di un computer.

chip Un circuito integrato che risiede su un piccolo quadratino di silicio montato su un contenitore plastico o ceramico.

ciclo Una sequenza di istruzioni di programma che vengono eseguite ripetutamente finchè si manifesta un certo evento, normalmente finchè una variabile non raggiunge un certo valore.

ciclo nidificato Un ciclo compreso dentro un altro ciclo.

ciclo senza fine Un ciclo che non ha un punto d'uscita e che viene eseguito per sempre. Questo è un errore comune fatto dai programmatori, quando non includono dentro il loop un test di condizione o quando il test è sempre vero (o sempre falso). L'interruzione di un ciclo senza fine richiede l'uso di uno speciale carattere di uscita, spesso è CTRL C.

circuito integrato Chiamato anche IC. Un circuito elettronico con molti transistor e funzioni logiche realizzato su un pezzetto di silicio.

codificazione L'azione di convertire un algoritmo o un diagramma di flusso in una sequenza di istruzioni di programma. È uno degli stadi del processo di programmazione.

comando Una parola riservata usata per eseguire una particolare mansione interna, come pulire lo schermo, far partire un programma o accedere ai file. L'introduzione di un comando attiva un programma specializzato interno al computer che esegue la mansione.

computer Un contenitore contenente almeno una unità di elaborazione (CPU), una memoria, delle interfacce di base che permettano di comunicare con l'esterno ed un alimentatore. Il contenitore può includere anche una tastiera, uno schermo e le unità a disco. Un computer è capace di memorizzre ed eseguire dei programmi. Comunica con il mondo esterno attraverso i dispositivi di ingresso/uscita. Il dispositivo d'ingresso tipico è la tastiera. Il dispositivo d'uscita usuale è lo schermo e probabilmente anche la stampante. Una memoria addizionale è generalmente disponibile sotto forma di unità a disco o registratore a cassette.

CPU *Central Processing Unit* (Unità centrale di elaborazione). Un modulo elettronico incaricato di prelevare, decodificare ed eseguire in una certa sequenza le istruzioni conservate nella memoria. Sulla maggior parte dei piccoli computer, la CPU e la memoria risiedono su una singola scheda o "card". La CPU usa normalmente un chip microprocessore e pochi altri componenti.

CRT Un tubo a raggi catodici. Uno schermo simile alla televisione.

cursore Un simbolo usato per indicare la posizione corrente sullo schermo nella quale verrà visualizzato il prossimo carattere. Spesso è un quadratino lampeggiante o una lineetta. Generalmente sono disponibili speciali tasti per posizionare convenientemente il cursore sullo schermo.

dati Testo o numeri sui quali opera il programma.

debugging L'eliminazione degli errori dal programma. Questa azione potrebbe essere ardua, perciò deve essere fatto ogni sforzo per scrivere un programma corretto che abbia meno errori possibile.

diagramma di flusso Rappresentazione visiva e simbolica di un algoritmo.

disco Mezzo magnetico sul quale vengono conservati dati e programmi. Sul disco, le informazioni sono organizzate a "files" che possono essere richiamate tramite un nome. I dischi possono memorizzare una grande quantità di dati e sono una comune memoria di massa usata nei piccoli computer.

disk drive Una *unità a disco.*

diskette Un floppy disk, cioè un disco flessibile da 8" o da 5"-1/4 progettato per una memorizzazione poco costosa di dati e programmi.

doppia precisione Un numero che ha una quantità doppia di cifre rispetto alla rappresentazione normale. Ogni interprete rappresenta i numeri con una certa quantità di cifre. La doppia precisione è richiesta generalmente per calcoli scientifici o applicazioni di affari che richiedano numeri molto grandi o molto precisi.

editor Un programma progettato per facilitare l'introduzione e la modifica di testi. Usato anche per eliminare gli errori o fare dei cambiamenti durante la scrittura dei programmi.

errore di programma Un *bug*.

espressione Una combinazione di operandi o variabili separati da operatori. Una espressione rappresenta una formula ed esegue un calcolo specifico. Quando una espressione viene calcolata dall'interprete, dà come risultato un valore.

espressione logica Una combinazione di operandi o variabili separati da operatori relazionali (=, > .etc.). Il valore di una espressione logica può essere vero o falso.

etichetta Nel BASIC, un numero di linea.

file Una raccolta di informazioni alla quale è stato attribuito un nome. Un programma viene normalmente memorizzato su disco come file.

floating-point number Un *numero a virgola mobile*.

flowchart Un *diagramma di flusso*.

grafica Immagini, figure o grafici visualizzati sullo schermo usando, generalmente, uno schema di puntini adiacenti uno all'altro. L'uso dei colori migliora l'aspetto dei grafici.

hardware L'apparecchiatura comprendente il computer, i dischi ed ogni altro elemento che forma un sistema computerizzato. *Opposto a:* software (cioè le istruzioni o i programmi).

IC Un *circuito integrato*.

Inizializzazione La fase (in un programma) durante la quale vengono assegnati i valori iniziali alle variabili. Ogni ciclo richiede una fase di inizializzazione.

interfaccia Un circuito elettronico che permette la connessione di un certo dispositivo al computer. Un disco, una stampante e un registratore richiedono una specifica interfaccia.

interprete Il programma incaricato di tradurre le istruzioni da un linguaggio programmativo (diciamo, il BASIC) nel linguaggio binario del computer ed eseguirlo. Una volta che l'interprete è stato collocato nel computer, il computer riesce a capire le istruzioni BASIC.

I/O *Input/Output* (ingresso/uscita) cioè la comunicazione da e per il computer.

istruzione Un ordine corretto dato all'interprete che avrà effetto sui dati sui quali si sta operando. Ogni istruzione preceduta da un numero di linea è parte di un programma. (I

comandi non hanno effetto sui valori dei dati. Essi eseguono delle mansioni interne o facilitano la scrittura o l'uso di un programma).

istruzione vuota Una istruzione che non fa niente. Per esempio, l'istruzione

10 REM

può essere usata per fare uno spazio nel programma e migliorarne la leggibilità.

K 1024, si legge "K" o "kilo". K è normalmente usato per rappresentare la dimensione della memoria generalmente in byte.

linguaggio ad alto livello Un linguaggio programmativo studiato per rendere più semplice dare le istruzioni ad un computer. Il BASIC è un linguaggio ad alto livello.

linguaggio macchina Il linguaggio binario che il computer comprende direttamente, cioè l'insieme limitato di istruzioni che manipolano l'informazione binaria nella CPU e nella memoria.

load Il caricamento.

logica Una espressione o una variabile che assume il valore vero o falso.

loop Un *ciclo.*

memoria Il mezzo che conserva le informazioni. La memoria interna risiede generalmente sulla stessa scheda della CPU e memorizza i dati ed i programmi sotto forma di byte. Unità di memoria di massa sono i dischi e le cassette.

microcomputer Un computer che usa un microprocessore come una unità centrale di elaborazione.

microprocessore Un circuito integrato che implementa la maggior parte delle funzioni di una CPU su un singolo chip. Un microprocessore odierno incorpora, di solito, decine di migliaia di transistor dentro un singolo chip e può anche incorporare la memoria.

monitor Il minimo programma necessario perchè un computer possa operare. Il monitor legge i caratteri dalla tastiera, li visualizza sullo schermo ed esegue i trasferimenti elementari di dati tra la tastiera, lo schermo e le comuni periferiche.

MPU *Microprocessor unit* (Unità a microprocessore). Un chip.

non-residente Un programma che normalmente non è nella memoria permanente (ROM) del computer. Un programma non-residente può essere conservato su una cassetta o su un dischetto.

numero a virgola fissa Un numero intero, cioè un numero con la virgola (punto decimale) in una posizione fissa alla destra dell'ultima cifra.

numero a virgola mobile Un numero decimale. Internamente vengono usate un numero fisso di cifre per rappresentare ogni numero e quando vengono eseguite le operazioni sul numero, la virgola (punto decimale) si muove a destra o sinistra.

operatore Un simbolo che rappresenta una valida operazione su un valore (cioè, + (addizione), * (moltiplicazione), etc.)

operatore relazionale Un operatore logico che stabilisce una relazione di grandezza tra due valori.

parola riservata Una parola che ha un significato predefinito per l'interprete BASIC. Non può essere usata dal programmatore come nome di variabile.

periferica Dispositivo collegato al computer, come una stampante, un disco o un terminale.

programma Sequenza di istruzioni che viene eseguita dal computer. Ogni programma è scritto in uno specifico linguaggio per computer e, per essere eseguito, deve essere caricato nella memoria del computer.

RAM *Random Access Memory* (memoria ad accesso casuale). La parte della memoria del computer (il resto è ROM) che può essere letta o scritta (cioè modificabile).

residente Un programma che è conservato permanentemente nella memoria del computer, cioè in ROM.

ROM *Read Only Memory* (memoria di sola lettura). Questa memoria non può essere cambiata dal programmatore. Generalmente contiene una parte o tutto il monitor e qualche volta un semplice interprete BASIC.

RUN Comando che fa partire l'esecuzione di un programma.

salto L'esecuzione di una istruzione al di fuori della normale sequenza.

sintassi Un insieme di regole che definiscono le istruzioni accettabili di un linguaggio programmativo. Il BASIC ha una sintassi semplice. L'interprete ricerca ed indica sempre gli errori di sintassi.

sistema operativo Un programma interno che fornisce delle facilitazioni per l'esecuzione di tutti i normali trasferimenti ed elaborazioni di dati richiesti per un conveniente uso delle risorse di un computer. Il sistema operativo organizza i file su disco, esegue conversioni di formato e fa partire ed arresta i programmi.

software I programmi.

statement Una istruzione BASIC o un comando che fa parte di un programma.

stringa Una sequenza di caratteri (*opposta a:* un numero). Nel BASIC, il nome di una variabile è differente a seconda che contenga una stringa o un numero.
Per esempio, PAROLA$ è una variabile stringa, mentre NUMERO è una variabile numerica.

terminale Combinazione di uno schermo e una tastiera o di una stampante e una tastiera, usata per comunicare convenientemente con un computer.

unità a disco Il meccanismo usato per leggere e scrivere su un disco.

variabile Locazione di memoria alla quale è stato assegnato un nome e che può contenere col tempo diversi valori. Il BASIC fa una distinzione tra le variabili numeriche e le variabili stringa. Il nome di una variabile stringa deve finire con $.

Indice analitico

Algoritmo, 141, 149
APL, 7
Assegnazione, istruzione di, 65, 68
Asterischi, linee di, 131
Asterisco, 79

Banca dati, 14
BASIC, 1, 2, 4-9
BASIC avanzato, 6
BASIC, espressione, 47
BASIC non residente, 5, 8
BASIC residente, 5, 8
BASIC, versioni del:
  avanzato, 6
  a virgola mobile, 6, 9, 42
  cassette, 6
  completo, 6
  dischi, 6
  esteso, 6, 8
  integer, 9
  mini, 8, 9
  tiny, 6, 8
  XBASIC, 24
Binaria, forma, 2
Bit, 2
BREAK (Interruzione), 19
Byte 2

Cancellare, 21
CAPS LOCK, 21
Caricamento, 7, 13
Cassette, BASIC, 6
Celsius, 50
Chiavi alfabetiche, 18
Chilometri, 50
Chip (circuito integrato), 12
Ciclo (Loop), 109, 121, 123, 127, 132
Circuito integrato (chip), 12
CLE, 83
CLS, 83
COBOL, 7
Codice di controllo (CTRL), 21
Codificazione, 141, 175
Collaudo, 158
Collaudo manuale, 154
Comandi, 33
Comandi di:
  CLE, 83
  CLS, 83
  END, 32, 34, 151
  esegui, 22
  EXIT, 21
  FOR NEXT, 126-130, 132, 135
  GOTO, 64, 108-114, 116, 121-124, 127, 178-180
  IF THEN, 64, 94, 96, 98, 99, 101, 102, 111,
  INPUT, 55, 56, 61, 63, 65-68, 71, 74, 77, 78, 80, 81, 84, 89, 90, 178-180
  LIST, 33, 34, 36, 89
  LPRINT, 26, 29
  NEW, 33, 35, 37, 39
  PRINT, 22, 31, 42, 47, 48, 84-86, 151, 160, 179-181
  RENUMBER, 89
  RUN, 27, 29, 33-35, 54, 55, 97, 98
  SAVE, 30, 32, 33
Compatibilità upwards, 7
Computer, 12
Consumo, 50
Contatore, 127
Controllo (CTRL), 18, 21
Controllo dei dati di ingresso, 115
t, 50
Falso, 94
Files, 9, 194
Forma binaria, 2
Formati, 47
FORTRAN, 7
Frecce, 150
Funzioni definite dall'utente, 192
Funzioni speciali, 19

Galloni, 50
GE 225, 7
GOTO, comando di, 64, 108-114, 116, 121-124, 127, 178-180
Grafico, 194

IF THEN, comando di, 64, 94, 96, 98, 99, 101, 102, III
Inizializzazione, 73, 113, 123, 127
Inizio, 151, 167, 169
Interruzione (BREAK), 19
INPUT, 54, 55, 57, 58, 61-63, 65-68, 71, 74, 78, 80, 81, 83, 86, 178
INPUT abbreviato, 84
INPUT, comando di, 55, 56, 61, 63, 65-68, 71, 74,
  77, 78, 80, 81, 84, 89, 90,
  178-180
INPUT, dispositivo di, 9
Interattivo, 7
Interfaccia, 12, 14
Interprete, 5, 14, 130
Istruzioni, 2, 3
Istruzioni, tipi di:
  assegnazione, 65, 68
  differita, 30
  eseguibili, 102
  FOR NEXT, 126-130, 132, 135, 189

Indice analitico

immediate, 30
INPUT, 68
LET, 65
multiple, 80
PRINT vuote, 84, 180
vuote, 37

K, 8
Kemeny John, 7, 8
Key Pad, 19, 22
Kurtz Thomas, 7

LET, istruzione di, 65
Linee di asterischi, 131
Lineette, 79
Linguaggi, 2
Linguaggi ad alto livello:
APL, 7
BASIC, 7
COBOL, 7
FORTRAN, 7
Pascal, 7
Linguaggio di programmazione ad alto livello, 4, 57
Linguaggio macchina, 2, 5
Linguaggio programmati o, 4
LIST, comando di, 33, 34, 36, 89
Listare, 26
LPRINT, comandi di, 26, 29
Loop (ciclo), 109, 121, 123, 127, 132

Maiuscolo, 20
Memoria, 12, 13
Menu, 104, 106
Messaggio di errore, 26
Microprocessore, 17
Mini BASIC, 8, 9
Modem, 14
Modo di calcolo, 29
Modo differito, 29
Modo normale, 29
Monitor, 14, 15

Neretto, dati in, 57
NEW, comando di, 33, 35, 37, 39
Nomi, 58
Nomi lunghi, 60
Nomi variabili, 55, 60, 86
Notazione scientifica, 43
Nozioni, 2
Numeri, 18
Numeri a virgola mobile, 6, 9, 42
Numeri di linea, 31

Operatore, 44
Operatore logico, 101
Operatore relazionale, 100
Operatore su stringa, 193
Operazioni, 44
Operazioni aritmetiche, 44, 47
Output, dispositivo di, 14

Parentesi, 46
Parola'chiave, 118
Parola riservata, 25, 60
Pascal, 7
Passo variabile, 132, 133
PRINT, comando di, 22, 31, 42, 47, 48, 84-86, 151,d
160, 179-181
PRINT vuoto, istruzione di, 84, 180
Progetto, algoritmo, 142, 166
Progetto, errori di, 159
Programma, 2, 3, 17, 26
Programmazione, 2
Programmazione ad alto livello, linguaggio di, 4, 5, 7
Prompt, 5, 24
Pulizia (CLE), 83
Punto e virgola, 48, 62

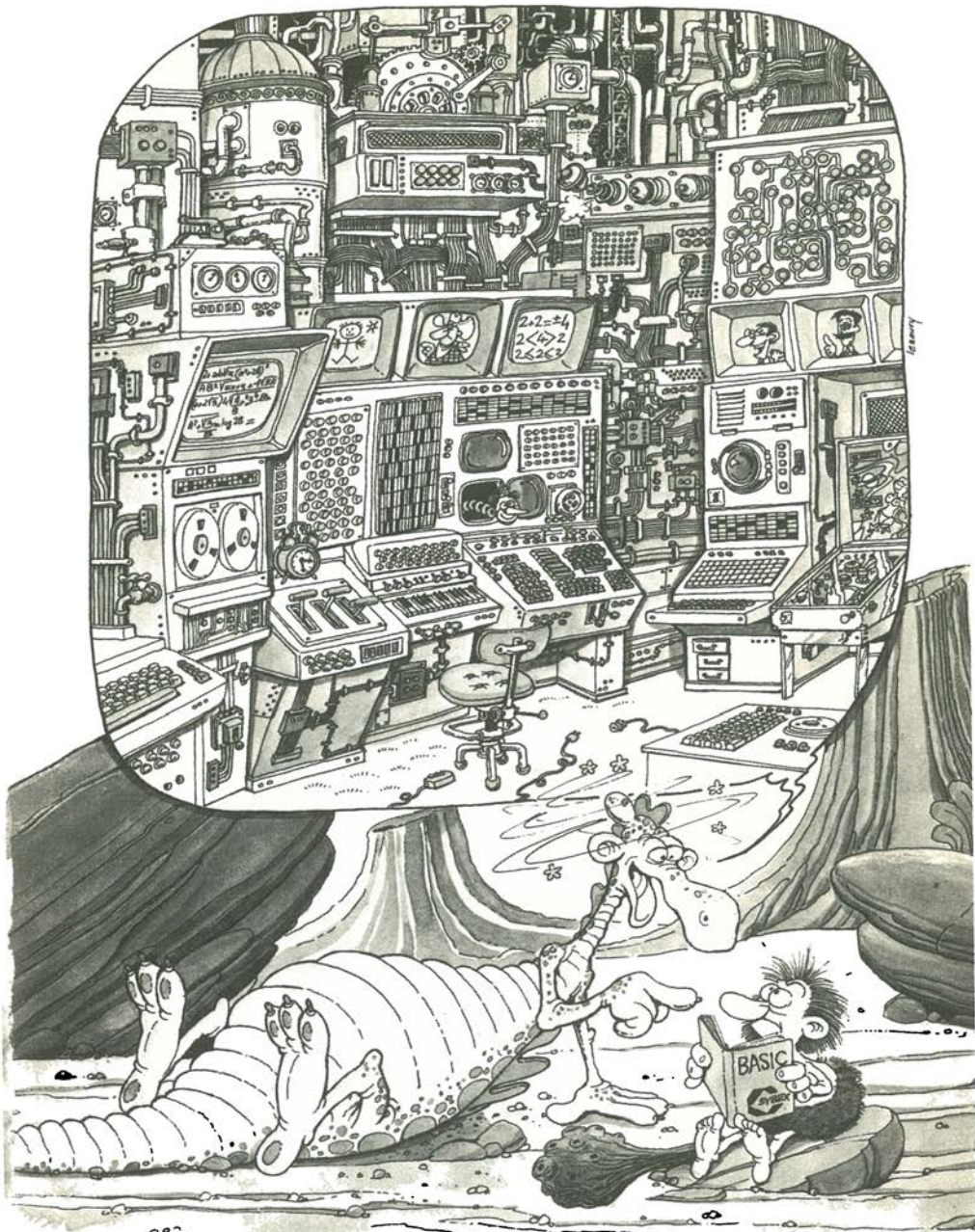
RAM, 13, 14, 32
Rappresentazione scientifica, 43
Read only memory (ROM), 13, 14
REM, 77-80, 82
RENUMBER, comando di, 89
RETURN, 20, 21, 24, 25, 26, 29, 54
Ritorno carrello (CR), 20, 21
ROM (Read only memory), 13, 14
RUBOUT, 19, 21
RUN, comando di, 27, 29, 33-35, 54, 55, 97, 98

Salto, 96, 180
SAVE, comando di, 32, 33
Schermo, 9, 10, 12
Segno uguale, 69
SHIFT, 20, 21
Simbolo, 4, 151
Sintassi, 4, 15, 68, 71
Somma dei primi N numeri interi, 129
Sottoprogramma (subroutine), 193
Spazio, fabbisogno di, 6
Spazio, 81, 82
Stampante, 14
Standard, 8
START, 151, 167, 169
Stringa, 42, 60
Stringa letterale, 64
Struttura dei dati, 194
Subroutine (sottoprogramma), 193

Tabella dei valori, 130
Tabulazione, 48
Tastiera, 9, 12, 17, 18
Tastierino numerico, 19, 22
Tecnica del contatore, 71, 112
Tecnica del IF/GOTO, 122
Test di validità, 177
Testi, 57
Testi, elaborazione dei, 2
Tipi di ciclo (loop):
— esterno, 137
— interno, 137
— nidificato, 135, 137
Tipi di variabili, 57
Tracciamento, 160
Troncato, 43

Unità a dischi, 14
Unità di elaborazione (CPU), 12
Utente, funzioni definite dall', 192

Valore:
— Celsius, 50
— esplicito, 68
— iniziale, 123
Variabile, 55
Variabile:
— intermedia, 67
— numerica, 58, 63
— stringa, 57, 60
Versioni, 8
Virgola, 47
Virgola mobile, BASIC a, 6, 9, 42
Virgolette, 48
Visualizzazione dei numeri, 42

2+2=±4
2<4>2
BASIC
SYBEX

Il pubblico che si interessa ai computer si è notevolmente allargato in questi ultimi anni e non si limita più a coloro che lo fanno con fini utilitari, pratici o di lavoro; esso si estende ora anche a coloro che vogliono conoscerne tutti gli aspetti. Conoscere la struttura e la logica di un computer e come si programma fa parte di questa esigenza. La diffusione del BASIC per la sua semplicità e quasi "naturalezza" di programmazione fa si che una cultura generale sull'informatica e la sua applicazione non può prescindere da una conoscenza di base di questo linguaggio. Questo lo scopo del libro: permettere anche a chi ha soltanto una cultura di base, di capire che cos'e il BASIC e come si usa.

L. 19.500

Cod. 507B ISBN 88-7056-182-8

162
Il tuo primo programma in BASIC
Rodnay Zaks
GRUPPO
EDITORIALE
JACKSON